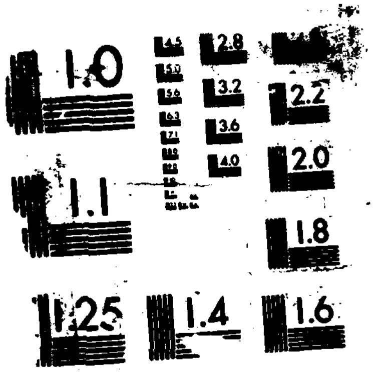
AD-A188 479   AN EFFICIENT NUMERICAL ALGORITHM FOR SOLVING SCATTERING   1/1
AND INVERSE SCATT (U) NUMERICAL COMPUTATION CORP STONY
BROOK NY   Y M CHEN 21 SEP 87 N00014-86-C-0109

UNCLASSIFIED   F/G 20/14   NL



MICROCOPY RESOLUTION TEST CHART

4

FINAL REPORT

AN EFFICIENT NUMERICAL ALGORITHM FOR SOLVING SCATTERING AND INVERSE SCATTERING PROBLEMS OF ELECTROMAGNETIC WAVES

PREPARED BY Y. M. CHEN
NUMERICAL COMPUTATION CORP.
57 QUAKER PATH
STONY BROOK, NEW YORK 11790-1309
PHONE: 516-751-9518

SEPTEMBER 21, 1987

APPROVED FOR PUBLIC RELEASE
DISTRIBUTION UNLIMITED

SBIR PHASE I CONTRACT NO. N00014-86-C-0109
OFFICE OF NAVAL RESEARCH
DEPARTMENT OF THE NAVY
800 N. QUINCY STREET
ARLINGTON, VA 22217-5000

SCIENTIFIC MONITOR:
DEFENSE ADVANCED RESEARCH PROJECTS AGENCY - TTO
1400 WILSON BLVD.
ARLINGTON, VA 22209

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION
UNCLASSIFIED

1b. RESTRICTIVE MARKINGS

2a. SECURITY CLASSIFICATION AUTHORITY
Directorate for Security Review, OASD(PA)

2b. DECLASSIFICATION / DOWNGRADING SCHEDULE

3. DISTRIBUTION / AVAILABILITY OF REPORT
APPROVED FOR PUBLIC RELEASE
DISTRIBUTION UNLIMITED

4. PERFORMING ORGANIZATION REPORT NUMBER(S)
FINAL REPORT

5. MONITORING ORGANIZATION REPORT NUMBER(S)
FINAL REPORT

6a. NAME OF PERFORMING ORGANIZATION
NUMERICAL COMPUTATION CORP.

6b. OFFICE SYMBOL *(If applicable)*

7a. NAME OF MONITORING ORGANIZATION
DARPA - TTO

6c. ADDRESS *(City, State, and ZIP Code)*
57 Quaker Path
Stony Brook, N.Y. 11790-1309

7b. ADDRESS *(City, State, and ZIP Code)*
1400 Wilson Blvd.
Arlington, VA 22209-2308

8a. NAME OF FUNDING / SPONSORING ORGANIZATION
Office of Naval Research

8b. OFFICE SYMBOL *(If applicable)*

9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
N00014-86-C-0109

8c. ADDRESS *(City, State, and ZIP Code)*
800 N. Quincy Street
Arlington, VA 22217-5000

10. SOURCE OF FUNDING NUMBERS

PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.	WORK UNIT ACCESSION N

11. TITLE *(Include Security Classification)*
An Efficient Numerical Algorithm for Solving Scattering and Inverse Scattering Problems of Electromagnetic Waves - Unclassified

12. PERSONAL AUTHOR(S)
Chen, Yung M.

13a. TYPE OF REPORT
Final

13b. TIME COVERED
FROM 85-12-15 TO 87-8-31

14. DATE OF REPORT *(Year, Month, Day)*
87-9-21

15. PAGE COUNT
44

16. SUPPLEMENTARY NOTATION

17. COSATI CODES

FIELD	GROUP	SUB-GROUP

18. SUBJECT TERMS *(Continue on reverse if necessary and identify by block number)*
Numerical, algorithm, scattering, electromagnetic, wave

19. ABSTRACT *(Continue on reverse if necessary and identify by block number)*

The development of an efficient numerical algorithm of determining the unknown material composition and shape of an arbitrary target (from the measured electromagnetic waves in the far field region will enhance the capability) of the defense radar system to defeat known evasive schemes. The first step in this research effort is the development of an efficient and versatile numerical algorithm for calculating the scattered electromagnetic waves/radar cross section by a target with known complex geometry and material property. Hence the purpose of this Phase I research is to develop an efficient numerical algorithm for solving two-dimensional scattering problems. This is achieved by using a special finite difference method based upon a natural spatial discretization of the integral form of Maxwell's equations on a non-orthogonal grid-system and the leap-frog finite differencing in the time domain. It has the advantages of being (a) more efficient than any other known numerical methods, (b) highly accurate due to the body-fitted grid system, and (c) the easiest numerical method to implement boundary conditions. The capability and feasibility of this two-dimensional computer code are tested by performing---

next page

20. DISTRIBUTION / AVAILABILITY OF ABSTRACT
☒ UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. □ DTIC USERS

21. ABSTRACT SECURITY CLASSIFICATION
Unclassified

22a. NAME OF RESPONSIBLE INDIVIDUAL
G. T. WINN, TIO

22b. TELEPHONE *(Include Area Code)*
202-694-5919

22c. OFFICE SYMBOL
DARPA

83 APR edition may be used until exhausted.
All other editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE

Block 19:

numerical simulations on few realistic examples, e.g., cylindrical objects with cross sections of a metallic jet and a composite airfoil. In these processes, the radar cross sections as functions of both the incident angle and the scattered angle are calculated and they seem to be quite good. Hence this Phase I research is completed successfully. KEYWORDS: RADAR CROSS SECTIONS, ELECTROMAGNETIC SCATTERING, [illegible]

SBIR PHASE I CONTRACT NO. N00014-86-C-0109
OFFICE OF NAVAL RESEARCH
DEPARTMENT OF THE NAVY
800 N. Quincy Street
Arlington, Virginia 22217-5000

TITLE: AN EFFICIENT NUMERICAL ALGORITHM FOR SOLVING SCATTERING AND INVERSE SCATTERING PROBLEMS OF ELECTROMAGNETIC WAVES

FINAL REPORT: Prepared by Y. M. Chen

NUMERICAL COMPUTATION CORP.
57 Quaker Path Ph. 516-751-9518
Stony Brook, N.Y. 11790-1309

DATE: September 21, 1987

TABLE OF CONTENTS

SUMMARY

The development of an efficient numerical algorithm capable of determining the unknown material composition and shape of an arbitrary target from the measured electromagnetic waves in the far field region will enhance the capability of the defense radar system to defeat known evasive schemes. The first step in this research effort is the development of an efficient and versatile numerical algorithm for calculating the scattered electromagnetic waves/radar cross section by a target with known complex geometry and material property. Hence the purpose of this Phase I research is to develop an efficient numerical algorithm for solving two-dimensional scattering problems. This is achieved by using a special finite difference method based upon a natural spatial discretization of the integral form of Maxwell's equations on a non-orthogonal grid-system and the leap-frog finite differencing in the time domain. It has the advantages of being (a) more efficient than any other known numerical methods, (b) highly accurate due to the body-fitted grid system, and (c) the easiest numerical method to implement boundary conditions. The capability and feasibility of this two-dimensional computer code are tested by performing numerical simulations on few realistic examples, e.g., cylindrical objects with cross sections of a metallic jet and a composite airfoil. In these processes, the radar cross sections as functions of both the incident angle and the scattered angle are calculated and they seem to be quite good. Hence this Phase I research is completed successfully.

DEGREE TO WHICH PHASE I OBJECTIVES HAVE BEEN MET

The purpose of the Phase I research is to develop an efficient numerical algorithm for solving two-dimensional scattering of electromagnetic waves by a target with complex geometry and material property. Now, we are happy to announce that the objectives of the Phase I research have been achieved completely and successfully. The complete 2-D computer code is given in the Appendix in this final report.

ANTICIPATED BENEFITS

As it stands by itself, the usefulness of the 2-D computer code developed here in Phase I research is rather limited, because in the real life there is no truly two-dimensional target. However, it has demonstrated beyond any doubt that the above developed numerical algorithm can be extended to compute efficiently and accurately the scattered E-M waves or radar cross section for the three-dimensional target with complex geometry and material property. The 3-D code will be an important subroutine of the Generalized Pulse-Spectrum Technique for solving the 3-D inverse scattering problems; moreover, it will be a very useful tool in advanced vulnerability analysis for any postulated target.

INTRODUCTION

The numerical methods for solving the electromagnetic waves scattering problems have a recent history of only two decades. Although the true and original radar scattering problem is a problem of solving the initial-boundary value problem of Maxwell's equations in the space-time domain, in earlier days most numerical methods are based upon first Fourier transforming the original problem in the space-time domain into a corresponding problem in the space-frequency domain and then solving it numerically for several chosen frequencies. In this way, some useful but incomplete scattering information can be extracted from these results without performing inverse Fourier transformation. In particular, the antique definition of the radar cross section was originally defined for a single-frequency source; its proportionality to the square of the electric field and the invalidity of the principle of superposition for this nonlinear situation will introduce further error into the calculation of radar cross section by this approach as compared to the true situation. Moreover, all numerical methods for solving the E-M scattering problem in the space-frequency domain can be reduced to the problem of solving a very large matrix equation (in particular the three-dimensional scattering problems) which requires an extra ordinarily large amount of CPU time and memory storage; thus it renders these methods inefficient.

An efficient finite difference method for solving E-M scattering problems in the space-time domain was first introduced by Yee [1] for the two-dimensional case and later applied to the three-dimensional case by Taflove and Brodwin [2], Holland [3], and Kunz and Lee [4]. This finite difference method requires a uniformly rectangular/cubical grid system and it is the most efficient method (see the review by Chen [5]). Unfortunately, for scatterers with curved boundaries, one needs extra ordinarily large amount of uniform rectangular/cubical grid zones to approximate the curved boundaries and minimize the undesirable "staircasing phenomenon", and thus it renders this numerical method inefficient in general.

Later, Mei, Cangellaris, Angelakos and Lin [6],[7] have presented the "Point-matched Time Domain Finite Element Method", a combination of the essential features of the standard Yee's finite difference method and the finite element method. Its efficiency is an improvement over the standard Yee's finite difference method due to its ability to compute on a body-fitted non-orthogonal grid system, but it is still not efficient enough due to the additional work in implementing any boundary conditions, e.g., extra interpolation and extrapolation are needed at the boundary grid zones.

Recently, Yee [8] has made a dramatic improvement of his method by applying his finite difference discretization in the most natural way to the integral form of Maxwell's equations on a general non-orthogonal grid system which makes the implementation of boundary conditions extremely easy. One can show that it is the most efficient (for the same accuracy) method available by performing the standard computational complexity analysis, i.e., to count the total floating point arithmetic operations needed in a typical calculation [5]. This algorithm can be vectorized and parallelized with great ease and hence it will be ideal for the vector and multi-processor computers.

Here in the Phase I research, Yee's improved method is generalized for solving the two-dimensional scattering problems of E-M waves by targets with complex geometry and material property. First, the whole space domain Ω is divided into three connected but non-overlapping sub-domains, the interior region Ω_1 representing the target and possessing a non-orthogonal cylindrical grid system centered in itself, the intermediate region Ω_2 representing the free space just outside of the target and possessing the same grid system, and the exterior region Ω_3 representing the far-field free space but truncated at a large distance away from the target and possessing the standard orthogonal cylindrical grid system (Fig. 1). Mathematically, an initial-boundary value problem of the integral form of Maxwell's equations must be solved numerically,

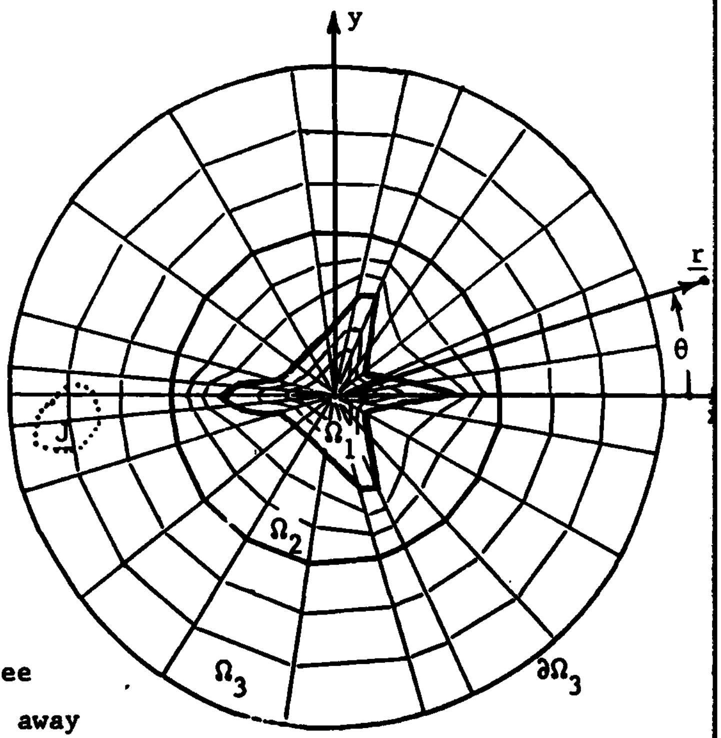


Fig. 1

$$\left.\begin{aligned} \oint \underline{E}_1 \cdot d\underline{\ell} &= -\oiint \underline{\underline{\mu}}\,\partial \underline{H}_1/\partial t \cdot d\underline{s},\\ \oint \underline{H}_1 \cdot d\underline{\ell} &= \oiint (\underline{\underline{\sigma}}\underline{E}_1 + \underline{\underline{\varepsilon}}\,\partial \underline{E}_1/\partial t) \cdot d\underline{s}, \end{aligned}\right. \quad \underline{x} \in \Omega_1, \tag{1}$$

$$\left.\begin{aligned} \oint \underline{E}_2 \cdot d\underline{\ell} &= -\oiint \mu_0\,\partial \underline{H}_2/\partial t \cdot d\underline{s},\\ \oint \underline{H}_2 \cdot d\underline{\ell} &= \oiint \varepsilon_0\,\partial \underline{E}_2/\partial t \cdot d\underline{s}, \end{aligned}\right. \quad \underline{x} \in \Omega_2, \tag{2}$$

$$\left.\begin{aligned} \oint \underline{E}_3 \cdot d\underline{\ell} &= -\oiint \mu_0\,\partial \underline{H}_3/\partial t \cdot d\underline{s},\\ \oint \underline{H}_3 \cdot d\underline{\ell} &= \oiint (\underline{J} + \varepsilon_0\,\partial \underline{E}_3/\partial t) \cdot d\underline{s}, \end{aligned}\right. \quad \underline{x} \in \Omega_3, \tag{3}$$

with boundary conditions (assuming no surface charges and currents),

$$\left.\begin{aligned} &\underline{n} \times \underline{E}_2 = \underline{n} \times \underline{E}_1, && \underline{n} \times \underline{H}_2 = \underline{n} \times \underline{H}_1,\\ &\varepsilon_0 \underline{E}_2 \cdot \underline{n} = \underline{\underline{\varepsilon}}\, \underline{E}_1 \cdot \underline{n}, && \mu_0 \underline{H}_2 \cdot \underline{n} = \underline{\underline{\mu}}\, \underline{H}_1 \cdot \underline{n}, \end{aligned}\right. \quad \underline{x} \in \partial\Omega_{12}, \tag{4}$$

and the asymptotic terminating condition,

$$\underline{n} \times \underline{E}_3 = (\mu_0/\varepsilon_0)^{1/2}(\underline{n} \times \underline{H}_3), \qquad \underline{x} \in \partial\Omega_3\,, \tag{5}$$

where $\underline{n}$ is the unit outer normal vector at the interfaces, $\partial\Omega_{12}$ is the interface between Ω_1 and Ω_2, $\partial\Omega_3$ is the outer boundary of Ω_3, $\underline{J}(\underline{x})$ is the source distribution, ε_0 and μ_0 are the free space permittivity and permeability respectively, $\underline{\underline{\varepsilon}}(\underline{x})$ is the 3 × 3 real positive symmetric permittivity matrix of the target, $\underline{\underline{\mu}}(\underline{x})$ is the 3 × 3 real positive symmetric permeability matrix of the target, and $\underline{\underline{\sigma}}(\underline{x})$ is the 3 × 3 real positive symmetric conductivity matrix of the target.

Since there exists no realistic two-dimensional E-M scattering problem, for the Phase I research the scattering of normal incident TEM electromagnetic wave by a cylindrical target with its axis along $\underline{i}_z$, e.g., $\underline{E} = E_x\underline{i}_x + E_y\underline{i}_y$ and $\underline{H} = H_z\underline{i}_z$, where $\underline{i}_a$ is the unit vector in the $\underline{a}$-direction.

FINITE DIFFERENCE METHOD

To discretize (1)-(3), the rectangle rule is used to approximate both the line and area integrals, the values of $\underline{E}$ and $\underline{H}$ fields are calculated on two different but staggered grid systems, and the leap-frog finite difference scheme is used to approximate the first order derivative in time.

Let each grid point of the basic non-orthogonal cylindrical grid system be denoted by $(r_{i,j}, \theta_j) \equiv (i,j)$, where "i" and "j" denote the i-th closed cylindrical grid line and the j-th radial grid line respectively. Let the center of the quadrilateral defined by (i,j), (i+1,j), (i,j+1) and (i+1,j+1) be denoted by $(i+\frac{1}{2},j+\frac{1}{2}) \equiv \{\frac{1}{4}(r_{i,j}+r_{i+1,j}+r_{i,j+1}+r_{i+1,j+1}), \frac{1}{2}(\theta_{j+1}+\theta_j)\}$, where all these centers form another non-orthogonal cylindrical grid system staggered on the basic grid system. Let the $\underline{E}$ fields be evaluated at the mid-points of the four edges of the quadrilaterals (Fig. 2) and at the integer time increments $t_n = n\Delta t$, n = 1,2,3,.....,; let the $\underline{H}$ fields be evaluated at the centers of the quadrilaterals (Fig. 2) and at the half time increments $t_{n+\frac{1}{2}} = (n+\frac{1}{2})\Delta t$, n = 0,1,2,3,..... For simplicity, let

$$\underline{\underline{\varepsilon}} = \begin{bmatrix} \varepsilon_x & 0 & 0 \\ 0 & \varepsilon_y & 0 \\ 0 & 0 & \varepsilon_z \end{bmatrix}, \quad \underline{\underline{\mu}} = \begin{bmatrix} \mu_x & 0 & 0 \\ 0 & \mu_y & 0 \\ 0 & 0 & \mu_z \end{bmatrix}, \quad \underline{\underline{\sigma}} = \sigma(\text{a scalar}),$$

and $\underline{J}$ be a point source located at the grid point (α,β). Let i = 1,2,3,...,I and j = 1,2,3,...,J.

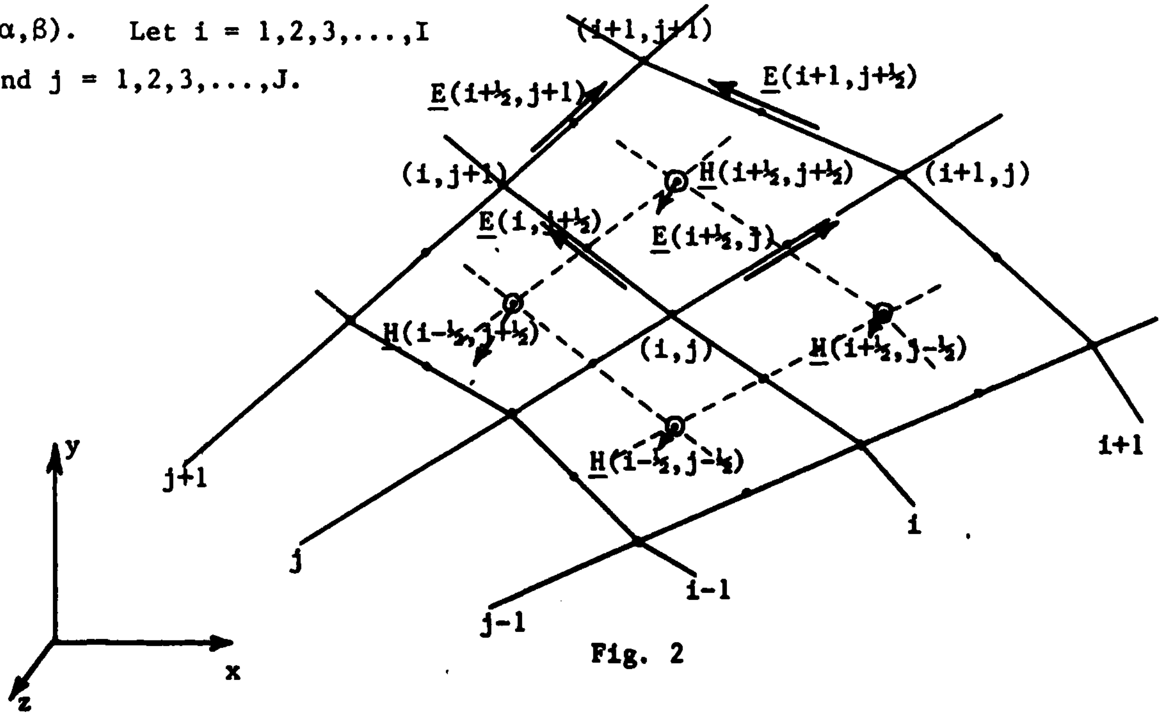

(i+1,j+1)
$\underline{E}(i+\frac{1}{2},j+1)$
$\underline{E}(i+1,j+\frac{1}{2})$
(i,j+1)
$\underline{H}(i+\frac{1}{2},j+\frac{1}{2})$
(i+1,j)
$\underline{E}(i,j+\frac{1}{2})$
$\underline{E}(i+\frac{1}{2},j)$
$\underline{H}(i-\frac{1}{2},j+\frac{1}{2})$
(i,j)
$\underline{H}(i+\frac{1}{2},j-\frac{1}{2})$
$\underline{H}(i-\frac{1}{2},j-\frac{1}{2})$
y
j+1
i+1
j
i
i-1
j-1
x
z

Fig. 2

The discretization of (1)-(3) in Ω_1 are,

$$H(\tfrac{1}{2},j+\tfrac{1}{2},n+\tfrac{1}{2}) = H(\tfrac{1}{2},j+\tfrac{1}{2},n-\tfrac{1}{2}) - \frac{2\Delta t}{\mu_z(\tfrac{1}{2},j+\tfrac{1}{2})(\theta_{j+1}-\theta_j)r_{1,j}r_{1,j+1}} \cdot \tag{6}$$

$$\{E(\tfrac{1}{2},j,n)r_{1,j} + E(1,j+\tfrac{1}{2},n)\delta\ell(1,j+\tfrac{1}{2}) - E(\tfrac{1}{2},j+1,n)r_{1,j+1}\},$$

$$\{\Delta t\sigma(\tfrac{1}{2},j)\delta\ell^2(\tfrac{1}{2},j) + \varepsilon_\lambda(\tfrac{1}{2},j)\}E(\tfrac{1}{2},j,n+1) \tag{7}$$

$$= \Delta t\sigma(\tfrac{1}{2},j)\{r_{\frac{1}{2},j+\frac{1}{2}}(1-(\theta_{j+1}-\theta_j)^2/8) - r_{\frac{1}{2},j-\frac{1}{2}}(1-(\theta_j-\theta_{j-1})^2/8)\}\delta\ell(\tfrac{1}{2},j)J^{-1}\sum_j E(\tfrac{1}{2},j,n)$$

$$+ \varepsilon_\lambda(\tfrac{1}{2},j)E(\tfrac{1}{2},j,n) + \Delta t\delta\ell^*(\tfrac{1}{2},j)(H(\tfrac{1}{2},j+\tfrac{1}{2},n+\tfrac{1}{2}) - H(\tfrac{1}{2},j-\tfrac{1}{2},n+\tfrac{1}{2})), \quad j = 1,2,3,\ldots,J,$$

$$n = 0,1,2,3,\ldots\ldots,$$

$$H(i+\tfrac{1}{2},j+\tfrac{1}{2},n+\tfrac{1}{2}) = H(i+\tfrac{1}{2},j+\tfrac{1}{2},n-\tfrac{1}{2}) \tag{8}$$

$$- \frac{\Delta t}{\mu_z(i+\tfrac{1}{2},j+\tfrac{1}{2})\delta A(i+\tfrac{1}{2},j+\tfrac{1}{2})}\{E(i+\tfrac{1}{2},j,n)(r_{i+1,j}-r_{i,j})$$

$$+ E(i+1,j+\tfrac{1}{2},n)\delta\ell(i+1,j+\tfrac{1}{2}) - E(i+\tfrac{1}{2},j+1,n)(r_{i+1,j+1}-r_{i,j+1})$$

$$- E(i,j+\tfrac{1}{2},n)\delta\ell(i,j+\tfrac{1}{2})\},$$

$$\{(\Delta t\sigma(i,j+\tfrac{1}{2})+ \varepsilon_\alpha(i,j+\tfrac{1}{2}))\delta\ell(i,j+\tfrac{1}{2}\ \}E(i,j+\tfrac{1}{2},n+1) \tag{9}$$

$$- \{\Delta t\sigma(i,j+\tfrac{1}{2})(r_{i,j+1}-r_{i,j})(1-(\theta_{j+1}-\theta_j)^2/8) + \varepsilon_\beta(i,j+\tfrac{1}{2})\}E(i+\tfrac{1}{2},j,n+1)$$

$$= \delta\ell(i,j+\tfrac{1}{2})\varepsilon_\alpha(i,j+\tfrac{1}{2})E(i,j+\tfrac{1}{2},n) - \varepsilon_\beta(i,j+\tfrac{1}{2})E(i+\tfrac{1}{2},j,n)$$

$$- \frac{\Delta t\ \delta\ell^*(i,j+\tfrac{1}{2})}{r_{i+\frac{1}{2},j+\frac{1}{2}}-r_{i-\frac{1}{2},j+\frac{1}{2}}}(H(i+\tfrac{1}{2},j+\tfrac{1}{2},n+\tfrac{1}{2}) - H(i-\tfrac{1}{2},j+\tfrac{1}{2},n+\tfrac{1}{2})),$$

$$-\{\Delta t\sigma(i+\tfrac{1}{2},j)[r_{i+\frac{1}{2},j+\frac{1}{2}}(1-(\theta_{j+1}-\theta_j)^2/8) - r_{i+\frac{1}{2},j-\frac{1}{2}}(1-(\theta_j-\theta_{j-1})^2/8)]+\varepsilon_\zeta(i+\tfrac{1}{2},j)\}\cdot \tag{10}$$

$$\delta\ell(i+\tfrac{1}{2},j)E(i,j+\tfrac{1}{2},n+1)$$

$$+ \{\Delta t\sigma(i+\tfrac{1}{2},j)\delta\ell^2(i+\tfrac{1}{2},j) + \varepsilon_\lambda(i+\tfrac{1}{2},j)\}E(i+\tfrac{1}{2},j,n+1)$$

$$= - \delta\ell(i+\tfrac{1}{2},j)\varepsilon_\zeta(i+\tfrac{1}{2},j)E(i,j+\tfrac{1}{2},n) + \varepsilon_\lambda(i+\tfrac{1}{2},j)E(i+\tfrac{1}{2},j,n)$$

$$+ \Delta t\delta\ell^*(i+\tfrac{1}{2},j)(H(i+\tfrac{1}{2},j+\tfrac{1}{2},n+\tfrac{1}{2}) - H(i+\tfrac{1}{2},j-\tfrac{1}{2},n+\tfrac{1}{2})),$$

$$j = 1,2,3,\ldots,J, \quad n = 0,1,2,3,\ldots\ldots$$

The discretization of (1)-(3) in Ω_2 are,

$$H(i+\tfrac{1}{2},j+\tfrac{1}{2},n+\tfrac{1}{2}) = H(i+\tfrac{1}{2},j+\tfrac{1}{2},n-\tfrac{1}{2}) \tag{11}$$

$$- \frac{\Delta t}{\mu_0\ A(i+\tfrac{1}{2},j+\tfrac{1}{2})}\{E(i+\tfrac{1}{2},j,n)(r_{i+1,j}-r_{i,j})+E(i+1,j+\tfrac{1}{2},n)\delta\ell(i+1,j+\tfrac{1}{2})$$

$$- E(i+\tfrac{1}{2},j+1,n)(r_{i+1,j+1}-r_{i,j+1}) - E(i,j+\tfrac{1}{2},n)\delta\ell(i,j+\tfrac{1}{2})\},$$

$$\{(\Delta t\sigma_0+\varepsilon_0)\delta\ell(i,j+\tfrac12)\}E(i,j+\tfrac12,n+1) \tag{12}$$
$$-(r_{i,j+1}-r_{i,j})(1-(\theta_{j+1}-\theta_j)^2/8)(\Delta t\sigma_0+\varepsilon_0)E(i+\tfrac12,j,n+1)$$
$$=\delta\ell(i,j+\tfrac12)\varepsilon_0E(i,j+\tfrac12,n)-\varepsilon_0(r_{i,j+1}-r_{i,j})(1-(\theta_{j+1}-\theta_j)^2/8)E(i+\tfrac12,j,n)$$
$$-\frac{\Delta t\delta\ell^*(i,j+\frac12)}{r_{i+\frac12,j+\frac12}-r_{i-\frac12,j+\frac12}}(H(i+\tfrac12,j+\tfrac12,n+\tfrac12)-H(i-\tfrac12,j+\tfrac12,n+\tfrac12)),$$

$$-(\Delta t\sigma_0+\varepsilon_0)\{r_{i+\frac12,j+\frac12}(1-(\theta_{j+1}-\theta_j)^2/8)-r_{i+\frac12,j-\frac12}(1-(\theta_j-\theta_{j-1})^2/8)\}\delta\ell(i+\tfrac12,j)E(i,j+\tfrac12,n+1)$$
$$+(\Delta t\sigma_0+\varepsilon_0)\delta\ell^2(i+\tfrac12,j)E(i+\tfrac12,j,n+1) \tag{13}$$
$$=-\varepsilon_0\{r_{i+\frac12,j+\frac12}(1-(\theta_{j+1}-\theta_j)^2/8)-r_{i+\frac12,j-\frac12}(1-(\theta_j-\theta_{j-1})^2/8)\}\delta\ell(i+\tfrac12,j)E(i,j+\tfrac12,n)$$
$$+\varepsilon_0\delta\ell^2(i+\tfrac12,j)E(i+\tfrac12,j,n)+\Delta t\delta\ell^*(i+\tfrac12,j)(H(i+\tfrac12,j+\tfrac12,n+\tfrac12)-H(i+\tfrac12,j-\tfrac12,n+\tfrac12))$$

$$j=1,2,3,\ldots,J,\quad n=0,1,2,3,\ldots..$$

The discretization of (1)-(3) in Ω_3 are,

$$H(i+\tfrac12,j+\tfrac12,n+\tfrac12)=H(i+\tfrac12,j+\tfrac12,n-\tfrac12) \tag{14}$$
$$-\frac{2\Delta t}{\mu_0(\theta_{j+1}-\theta_j)(r_{i+1}-r_i)(r_{i+1}+r_i)}\{(E(i+\tfrac12,j,n)-E(i+\tfrac12,j+1,n))(r_{i+1}-r_i)$$
$$+(r_{i+1}E(i+1,j+\tfrac12,n)-r_iE(i,j+\tfrac12,n))(\theta_{j+1}-\theta_j)\},$$

$$E(i,j+\tfrac12,n+1)=(\Delta t\sigma_0+\varepsilon_0)^{-1}\{\varepsilon_0E(i,j+\tfrac12,n)-\frac{2\Delta t}{(r_{i+1}-r_{i-1})}(H(i+\tfrac12,j+\tfrac12,n+\tfrac12)-H(i-\tfrac12,j+\tfrac12,n+\tfrac12))$$
$$-\delta_{i,\alpha}\delta_{j+\frac12,\beta}\Delta t\ (-J_x(\alpha,\beta,n+\tfrac12)\sin\tfrac12(\theta_{j+1}+\theta_j) \tag{15}$$
$$+J_y(\alpha,\beta,n+\tfrac12)\cos\tfrac12(\theta_{j+1}+\theta_j))\},$$

$$E(i+\tfrac12,j,n+1)=(\Delta t\sigma_0+\varepsilon_0)^{-1}\{\varepsilon_0E(i+\tfrac12,j,n)+\frac{4\Delta t(H(i+\frac12,j+\frac12,n+\frac12)-H(i+\frac12,j-\frac12,n+\frac12))}{(r_{i+1}+r_i)(\theta_{j+1}-\theta_{j-1})}$$
$$-\delta_{i,\alpha}\delta_{j+\frac12,\beta}\Delta t[-J_x(\alpha,\beta,n+\tfrac12)(\sin\tfrac12(\theta_{j+1}+\theta_j)-\sin\tfrac12(\theta_j+\theta_{j-1}))$$
$$+J_y(\alpha,\beta,n+\tfrac12)(\cos\tfrac12(\theta_{j+1}+\theta_j)-\cos\tfrac12(\theta_j+\theta_{j-1}))]\}, \tag{16}$$

$j=1,2,3,\ldots,J,\quad n=0,1,2,3,\ldots..,$

where $\delta A(i+\tfrac12,j+\tfrac12)=\tfrac12(\theta_{j+1}-\theta_j)\{(r_{i+1,j}-r_{i,j})r_{i,j+1}+(r_{i+1,j+1}-r_{i,j+1})r_{i+1,j}\},$

$\delta\ell^*(i,j+\tfrac12)=\tfrac12(r_{i,j+1}+r_{i,j})(\theta_{j+1}-\theta_j),$

$\delta\ell^*(i+\tfrac12,j)=\tfrac12\{r_{i+\frac12,j+\frac12}(\theta_{j+1}-\theta_j)+r_{i+\frac12,j-\frac12}(\theta_j-\theta_{j-1})\},$

$r_{i+\frac12,j+\frac12}=\tfrac14(r_{i,j}+r_{i,j+1}+r_{i+1,j}+r_{i+1,j+1}),$

$r_{i+\frac12,j-\frac12}=\tfrac14(r_{i,j-1}+r_{i,j}+r_{i+1,j}+r_{i+1,j}),$

$$
\begin{aligned}
\delta\ell(i,j+\tfrac{1}{2}) &= \{(r_{i,j+1}-r_{i,j})^2 + r_{i,j}r_{i,j+1}(\theta_{j+1}-\theta_j)^2\}^{\frac{1}{2}},\\
\delta\ell(i+\tfrac{1}{2},j) &= \{(r_{i+\frac{1}{2},j+\frac{1}{2}}-r_{i+\frac{1}{2},j-\frac{1}{2}})^2 + \tfrac{1}{4}r_{i+\frac{1}{2},j+\frac{1}{2}}r_{i+\frac{1}{2},j-\frac{1}{2}}(\theta_{j+1}-\theta_{j-1})^2\}^{\frac{1}{2}},\\
\varepsilon_\alpha(i,j+\tfrac{1}{2}) &= \varepsilon_x(i,j+\tfrac{1}{2})\sin^2\tfrac{1}{2}(\theta_{j+1}+\theta_j) + \varepsilon_y(i,j+\tfrac{1}{2})\cos^2\tfrac{1}{2}(\theta_{j+1}+\theta_j),\\
\varepsilon_\beta(i,j+\tfrac{1}{2}) &= \varepsilon_x(i,j+\tfrac{1}{2})(r_{i,j+1}\sin\theta_{j+1} - r_{i,j}\sin\theta_j)\sin\tfrac{1}{2}(\theta_{j+1}+\theta_j)\\
&\quad + \varepsilon_y(i,j+\tfrac{1}{2})(r_{i,j+1}\cos\theta_{j+1} - r_{i,j}\cos\theta_j)\cos\tfrac{1}{2}(\theta_{j+1}+\theta_j),\\
\varepsilon_\lambda(i+\tfrac{1}{2},j) &= \varepsilon_x(i+\tfrac{1}{2},j)\{r^2_{i+\frac{1}{2},j+\frac{1}{2}}\sin^2\tfrac{1}{2}(\theta_{j+1}+\theta_j) + r^2_{i+\frac{1}{2},j-\frac{1}{2}}\sin^2\tfrac{1}{2}(\theta_j+\theta_{j-1})\\
&\qquad -2r_{i+\frac{1}{2},j+\frac{1}{2}}r_{i+\frac{1}{2},j-\frac{1}{2}}\sin\tfrac{1}{2}(\theta_{j+1}+\theta_j)\sin\tfrac{1}{2}(\theta_j+\theta_{j-1})\}\\
&\quad + \varepsilon_y(i+\tfrac{1}{2},j)\{r^2_{i+\frac{1}{2},j+\frac{1}{2}}\cos^2\tfrac{1}{2}(\theta_{j+1}+\theta_j) + r^2_{i+\frac{1}{2},j-\frac{1}{2}}\cos^2\tfrac{1}{2}(\theta_j+\theta_{j-1})\\
&\qquad -2r_{i+\frac{1}{2},j+\frac{1}{2}}r_{i+\frac{1}{2},j-\frac{1}{2}}\cos\tfrac{1}{2}(\theta_{j+1}+\theta_j)\cos\tfrac{1}{2}(\theta_j+\theta_{j-1})\},\\
\varepsilon_\zeta(i+\tfrac{1}{2},j) &= \varepsilon_x(i+\tfrac{1}{2},j)\sin\theta_j(r_{i+\frac{1}{2},j+\frac{1}{2}}\sin\tfrac{1}{2}(\theta_{j+1}+\theta_j) - r_{i+\frac{1}{2},j-\frac{1}{2}}\sin\tfrac{1}{2}(\theta_j+\theta_{j-1}))\\
&\quad + \varepsilon_y(i+\tfrac{1}{2},j)\cos\theta_j(r_{i+\frac{1}{2},j+\frac{1}{2}}\cos\tfrac{1}{2}(\theta_{j+1}+\theta_j) - r_{i+\frac{1}{2},j-\frac{1}{2}}\cos\tfrac{1}{2}(\theta_j+\theta_{j-1})),\\
\varepsilon_x(i,j+\tfrac{1}{2}) &= \tfrac{1}{2}(\varepsilon_x(i+\tfrac{1}{2},j+\tfrac{1}{2}) + \varepsilon_x(i-\tfrac{1}{2},j+\tfrac{1}{2})),\\
\varepsilon_y(i,j+\tfrac{1}{2}) &= \tfrac{1}{2}(\varepsilon_y(i+\tfrac{1}{2},j+\tfrac{1}{2}) + \varepsilon_y(i-\tfrac{1}{2},j+\tfrac{1}{2})),\\
\varepsilon_x(i+\tfrac{1}{2},j) &= \tfrac{1}{2}(\varepsilon_x(i+\tfrac{1}{2},j+\tfrac{1}{2}) + \varepsilon_x(i+\tfrac{1}{2},j-\tfrac{1}{2})),\\
\varepsilon_y(i+\tfrac{1}{2},j) &= \tfrac{1}{2}(\varepsilon_y(i+\tfrac{1}{2},j+\tfrac{1}{2}) + \varepsilon_y(i+\tfrac{1}{2},j-\tfrac{1}{2})).
\end{aligned}
$$

Theoretically, the boundary conditions (4) must be imposed at the interface of two different materials. But here, there is no need to impose the boundary conditions explicitly in programming this numerical algorithm, because the boundary condition for the tangential component of $\underline{E}$ is satisfied automatically, and the other three boundary conditions are also automatically satisfied in the approximate sense if the differences of the material properties spread linearly across a complete grid zone instead of just across the interface. In this way, there is no cumbersome programming instruction at the interface to slow down the calculation and the application of boundary conditions is replaced by the process of assigning the material parameters into the grid zones. In particular, the programming instruction is extremely simple if the material interface is located either at a constant i-line or at a constant j-line.

At the exterior boundary $\partial\Omega_3$ $(i = I)$, the following simple but effective discretization of the asymptotic terminating of non-reflecting condition (5) is used,

$$E(I,j+\tfrac{1}{2},n+1) = (\mu_0/\varepsilon_0)^{\frac{1}{2}} H(I-\tfrac{1}{2},j+\tfrac{1}{2},n+\tfrac{1}{2}), \qquad (17)$$

$$j = 1,2,3,\ldots,J, \quad n = 0,1,2,3,\ldots..$$

Finally, to achieve a stable computation for this explicit finite difference scheme, one must impose the following well known stability condition,

$$\Delta t < \underset{i,j}{\text{Min.}}\ \delta\ell_{i,j} \cdot \underset{i,j,x,y,z}{\text{Min.}}\ \varepsilon_{x,y,z}(i+\tfrac{1}{2},j+\tfrac{1}{2}) \cdot \underset{i,j,x,y,z}{\text{Min.}}\ \mu_{x,y,z}(i,j), \qquad (18)$$

where $\delta\ell_{i,j}$ is the typical dimension of the i-jth grid zone.

NUMERICAL SIMULATION

From Eqs. (6)-(16), the discrete values of the E-M fields in the space-time domain scattered by a two-dimensional target with complex geometry and material property can be calculated as accurate as one desires. However, the main interest for the radar scattering application is not the detail description of the E-M fields everywhere; rather it is a certain average quantity to characterize the scattering and absorbing properties of a scatterer in the incident E-M fields. The quantity mostly common used for this purpose is the radar cross section which initially was defined for the time-harmonic E-M fields [9] as a normalized scattered power intensity averaged over a cycle or a normalized scattered energy intensity per cycle,

$$\Sigma(\psi) = \lim_{r\to\infty} \Gamma \frac{|\underline{E}_s|^2}{|\underline{E}_{in}^{o}|^2}, \qquad (19)$$

where $\Gamma = 2\pi r$ for two-dimensional problems,

$= 4\pi r^2$ for three-dimensional problems,

r = distance between the transmitter and the scatterer,

ψ = angle between the transmitter and the zero angle axis,

$\underline{E}_{in}^{o}$ = incident E-field at the target,

and $\underline{E}_s$ = scattered far field E-field.

Since the antique definition of the radar cross section (19) was defined only for a single-frequency source, errors will be introduced when it is applied to the scattering problems with general time-varying sources. In order to be able to deal with more realistic situations, the definition of the radar cross section (19) is generalized to cases with C-W pulse as source as the normalized scattered energy intensity,

$$\Sigma(\psi) = \lim_{r\to\infty} \Gamma \frac{\int_{T_s}^{T_s+\Delta T_s} |\underline{E}_s|^2 \, dt}{\int_{T_{in}}^{T_{in}+\Delta T_{in}} |\underline{E}_{in}^o|^2 \, dt} \quad , \tag{20}$$

where ΔT_{in} is the duration of the incident E-M pulse and ΔT_s is the duration of the scattered E-M pulse, i.e., $\underline{E}_{in}^o \equiv 0$ for $t < T_{in}$ and $t > T_{in}+\Delta T_{in}$, and $\underline{E}_s \equiv 0$ for $t < T_s$ and $t > T_s+\Delta T_s$.
It is clear that the definition (19) is a special case of the definition (20).

Now, there are two approaches to calculate the radar cross section of a given target. One approach is first to approximate the time-harmonic source by a C-W pulse with very long duration, next one runs the above developed computer code long enough for all of the transients to died off, and finally the definition (19) is used to compute the radar cross section. The other approach is to use a very short C-W pulse as the source, next the computer code is run long enough for the scattered C-W pulse passing through the location of the receiver completely, and finally the definition (20) is used to compute the radar cross section. It is obvious that the cost for the first approach to calculate the radar cross section is prohibitively high and the second approach is the preferred one.

To achieve our goal, a simple and almost costless sub-routine is added to the computer code for calculating $\int_0^t |\underline{E}(\underline{x},\tau)|^2 d\tau$ in the center of every grid zone at the outer boundary $\partial\Omega_3$, i.e.,

$$\sum_{m=1}^{n} |\underline{E}(I-\tfrac{1}{2},j+\tfrac{1}{2},m)|^2 \Delta t, \quad j = 1,2,3,\ldots,J, \quad n = 1,2,3,\ldots\ldots,$$

are calculated in the code; in general, it is a monotonic non-decreasing function of t.

In the region where the durations of $\underline{E}_{in}$ and $\underline{E}_s$ pulses are distinctively non-overlapping ($\psi < 100^o$), the value of the energy integral rises to a

constant value $\int_0^{T_{in}+\Delta T_{in}} |\underline{E}(\underline{x},\tau)|^2 d\tau$ after the passing through of the the $\underline{E}_{in}$ pulse and later it will rise again to another constant value $\int_0^{T_s+\Delta T_s} |\underline{E}(\underline{x},\tau)|^2 d\tau$ after the passing through of the $\underline{E}_s$ pulse. Then the value of $\int_{T_s}^{T_s+\Delta T_s} |\underline{E}_s|^2 d\tau$ can be obtained simply from $\int_0^{T_s+\Delta T_s} |\underline{E}|^2 d\tau - \int_0^{T_{in}+\Delta T_{in}} |\underline{E}|^2 d\tau$. A schematic diagram of this is given in Fig. 3.
As for the value of $\int_{T_{in}}^{T_{in}+\Delta T_{in}} |\underline{E}_{in}^o|^2 d\tau$, it can be obtained as $\int_0^{T_{in}+\Delta T_{in}} |\underline{E}_{in}|^2 d\tau$ at the target location with the target replaced by the free space in a separate calculation.

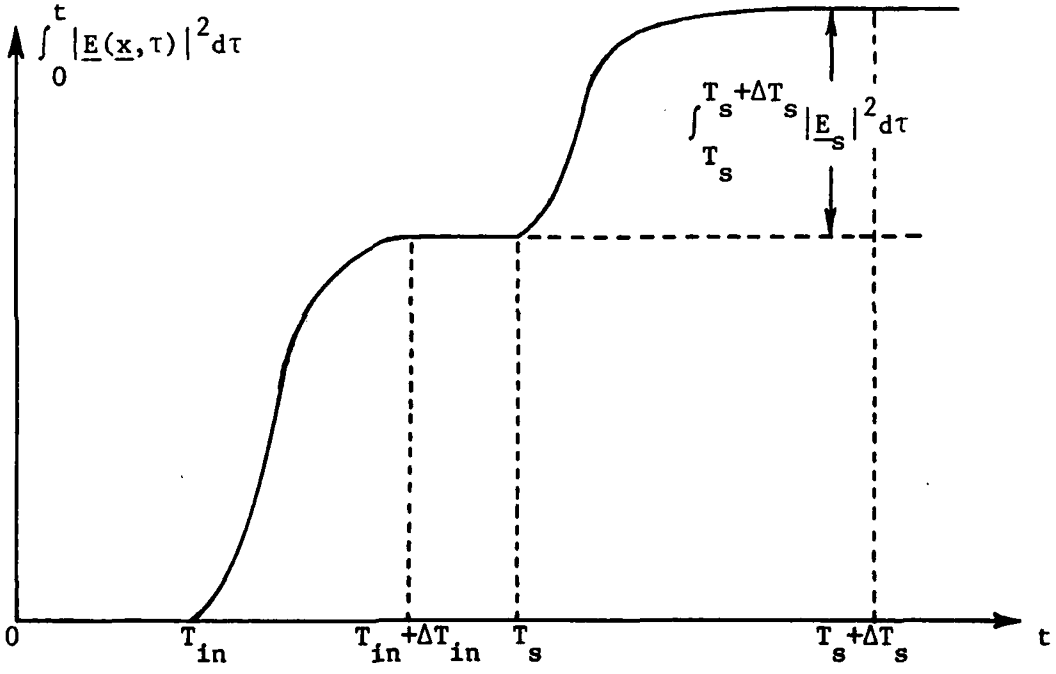


Fig. 3

As Example one, a jet consisting of perfectly conducting material with a characteristic dimension ∿ 10 m is used as the target. The computational grid system is shown in Fig. 4 with the radial grid size $\Delta r = 1$ m in Ω_3. The computational time increment Δt is chosen to be 1.25 ns. A dipole point source with Gaussian distribution in time and a duration of $10\Delta t$ is placed in Ω_3 with the location marked by "x" in Fig. 4. This current source generates a short C-W pulse of E-M fields with frequency ∿ 40 MHz (wavelength ∿ 7.5 m). The numerical results of radar cross sections $\Sigma(6^o)$, $\Sigma(16^o)$, $\Sigma(35^o)$, $\Sigma(62.5^o)$, and $\Sigma(88.5^{\circ})$ as functions of θ are plotted in Figs. 4, 5, 6, 7, and 8, respectively. Moreover, the radar back-scattering cross section as a function of $\theta(0 \leq \theta < 90^{\circ})$ is plotted in Fig. 9; since it is symmetric with respect to $\theta = 0$, its values in the range of $-90^{\circ} < \theta \leq 0$ are omitted in Fig. 9. As a matter of fact, the newly developed computer code in this project is used to calculate the values of the radar back-scattering cross section in the range of negative θ and it is found that the maximum deviation between its corresponding values at either side of $\theta = 0$ is less than 3.5%.

As Example two, a composite airfoil with the leading edge consisting of anisotropic lossy dielectric, $\varepsilon_x = 22.989\times10^{-12}$farad/m, $\varepsilon_y = 44.210\times10^{-12}$farad/m and $\sigma = 5\times10^{2}$mho/m, and the trailing edge consisting of perfectly conducting material is used as the target. The characteristic length of the airfoil is 10 m. The computational grid system is shown in Fig. 10 and the computational time increment Δt is chosen to be 0.15 ns. A dipole point source with Gaussian distribution in time and a duration of 2.1 ns is placed at $(14\text{m},\psi)$. This current source generates a short C-W pulse of E-M fields with frequency $\sim$ 238 MHz (wavelength $\sim$ 1.26m). The numerical results of radar cross sections $\Sigma(32.5^{\circ})$, $\Sigma(22.5^{\circ})$, $\Sigma(12.5^{\circ})$ and $\Sigma(2.5^{\circ})$ as functions of θ are plotted in Figs. 11, 12, 13, and 14 respectively. Similarly, the radar back-scattering cross section as a function of positive θ is plotted in Fig. 15; again due to the symmetry, the radar back-scattering cross sections for the negative values of θ are omitted here.

TECHNICAL DISCUSSION

Obviously, the numerical results for the radar cross section obtained from the calculation in the previous section are correct only in a very approximate sense. The main reason is that the supposedly scattered far field $\underline{E}_s$ is computed at the location about few wavelengths away from the target surface (actually in the intermediate field region), an error in physical representation. The simplest way to improve the accuracy is to add many radial zones in Ω_3 so that the boundary $\partial\Omega_3$ will be in the truly far field region. Unfortunately, this approach will not really solve the accuracy problem, because the ratio of the maximum dimension to the minimum dimension of the grid zone in the far field region will be so enormous that it will introduce large numerical errors.

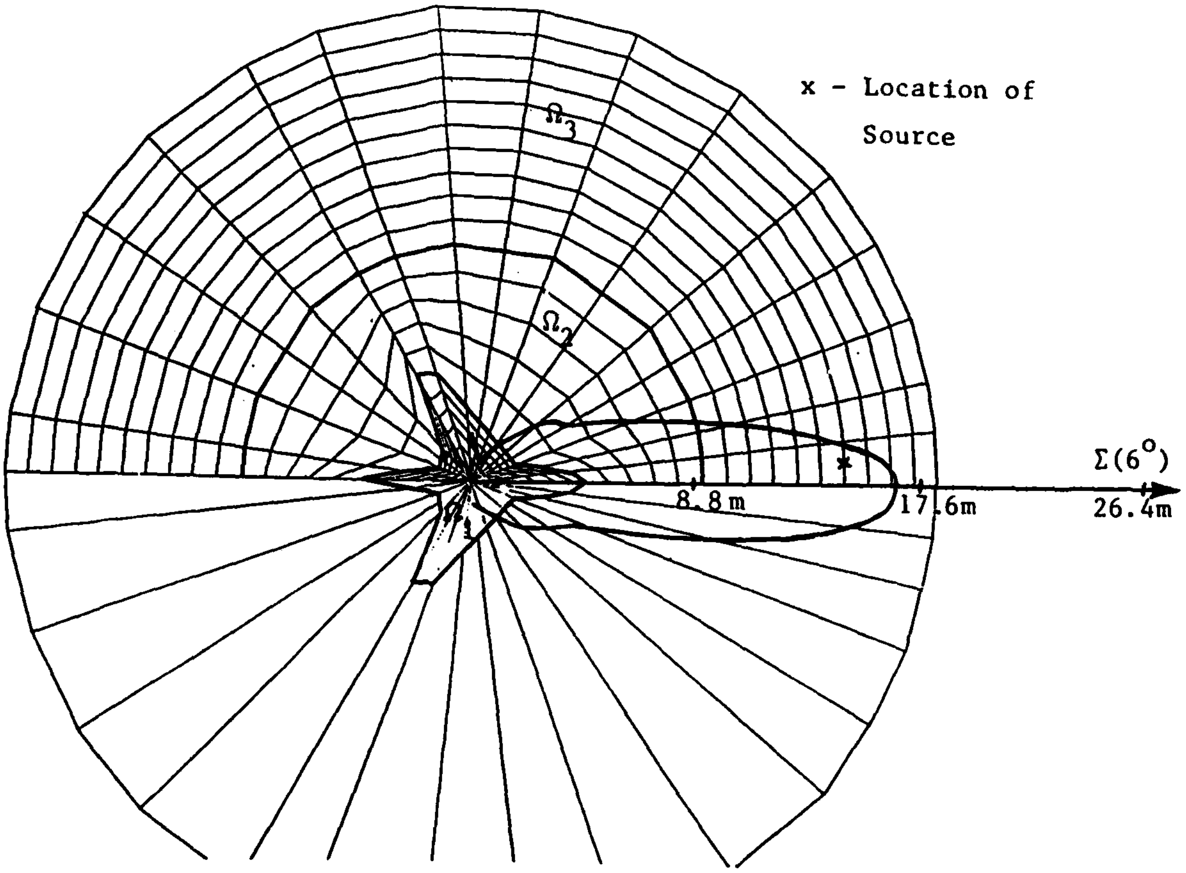

x - Location of Source
Ω3
Ω2
Ω1
Σ(6°)
8.8m
17.6m
26.4m

Fig. 4

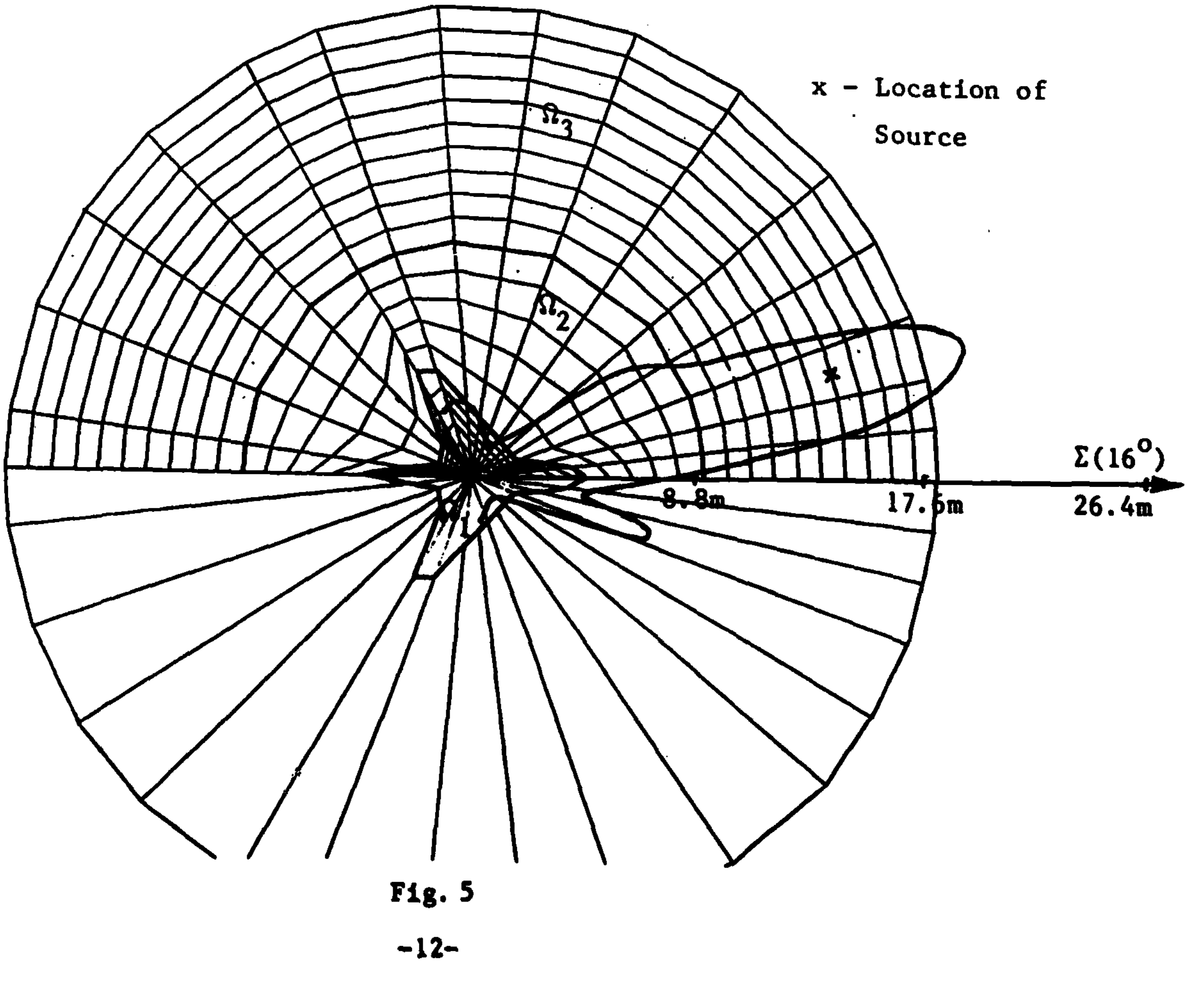

x - Location of Source
Ω3
Ω2
Ω1
Σ(16°)
8.8m
17.6m
26.4m

Fig. 5

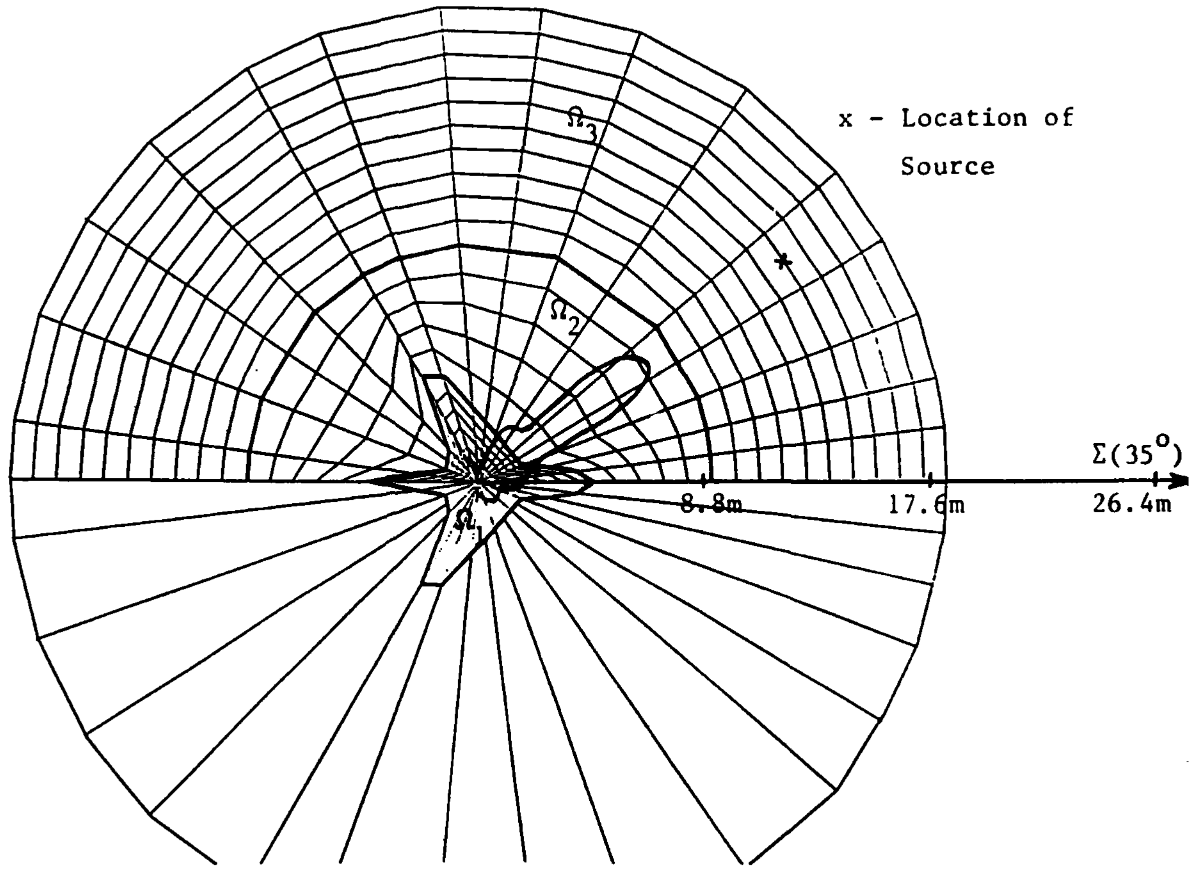

x - Location of
Source
Ω_3
Ω_2
Ω_1
$\Sigma(35^o)$
8.8m
17.6m
26.4m

Fig. 6

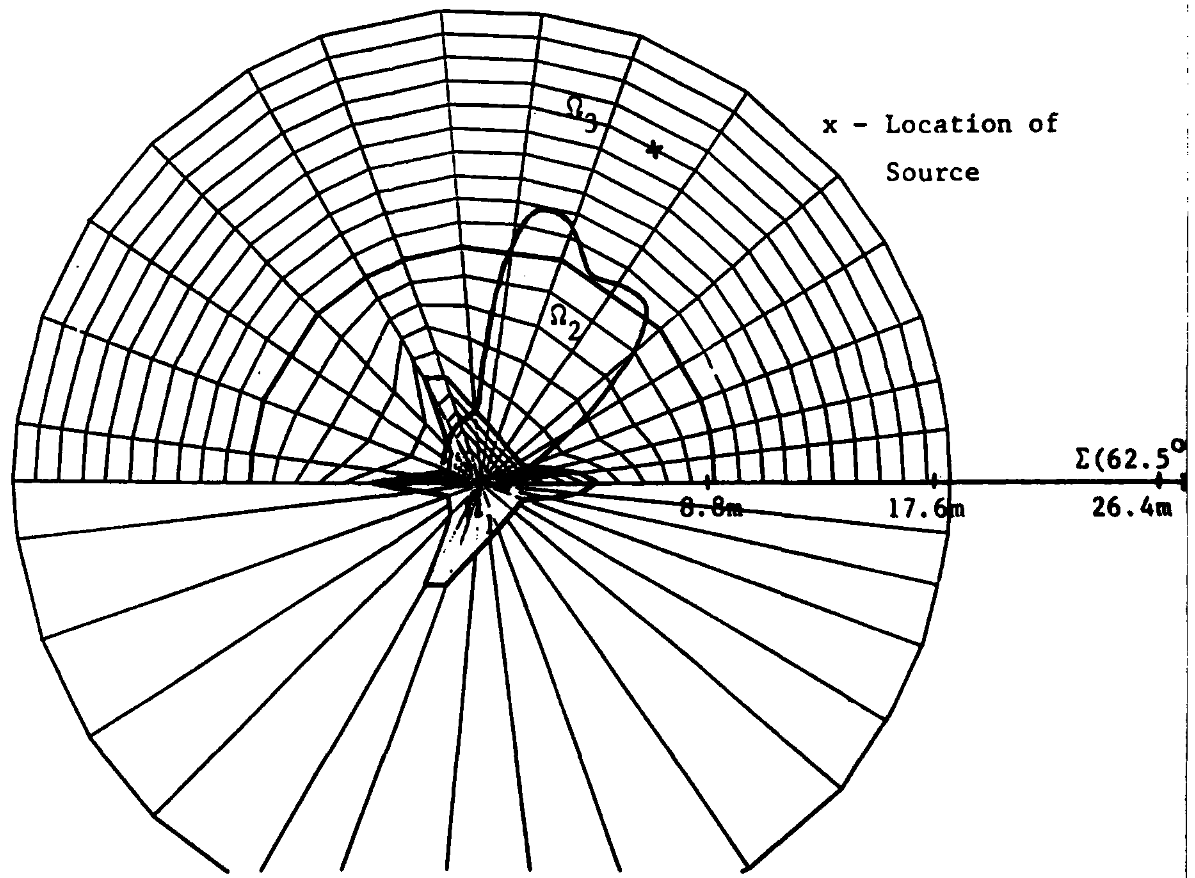

x - Location of
Source
Ω_3
Ω_2
$\Sigma(62.5^o$
8.8m
17.6m
26.4m

Fig. 7

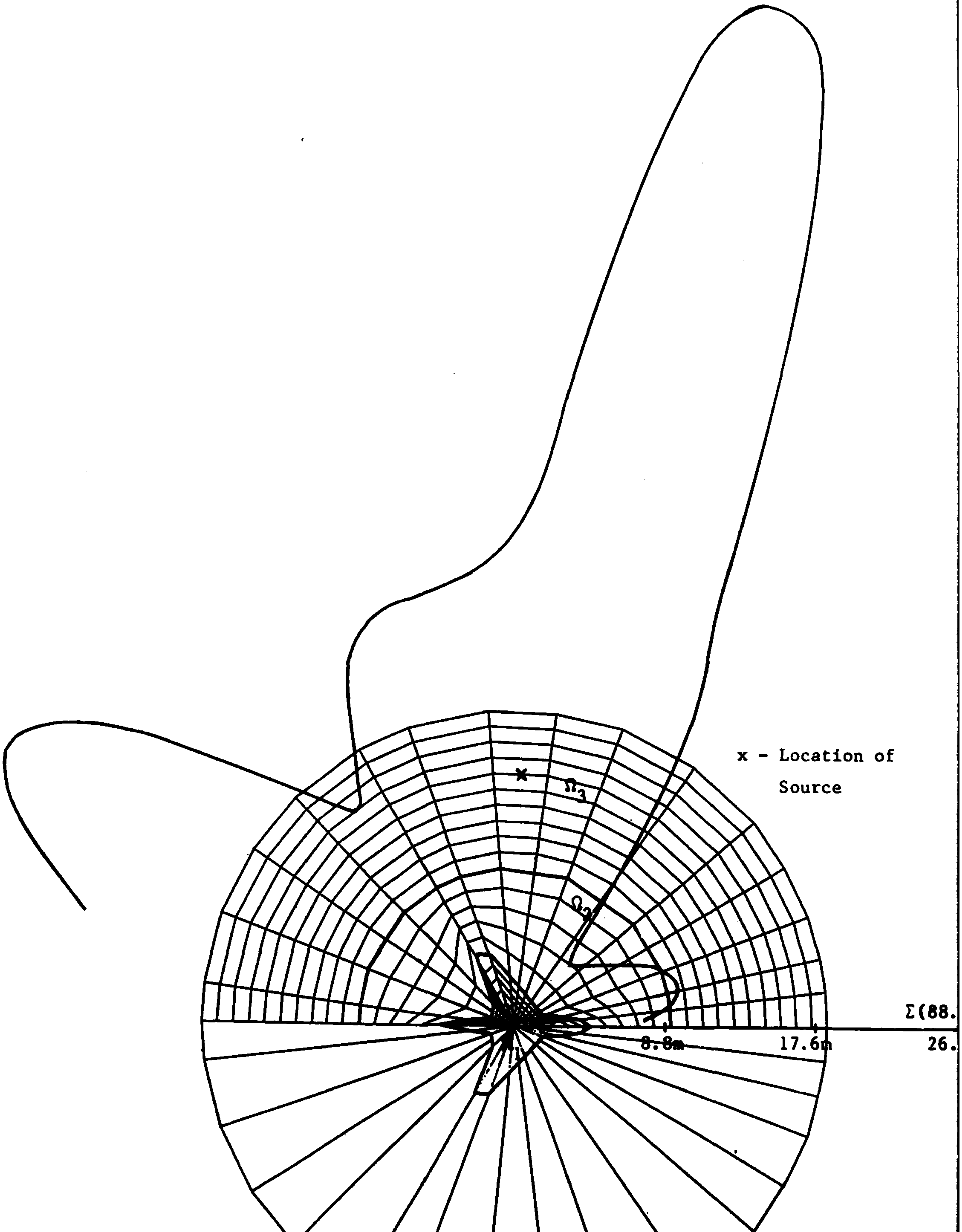


Fig. 8

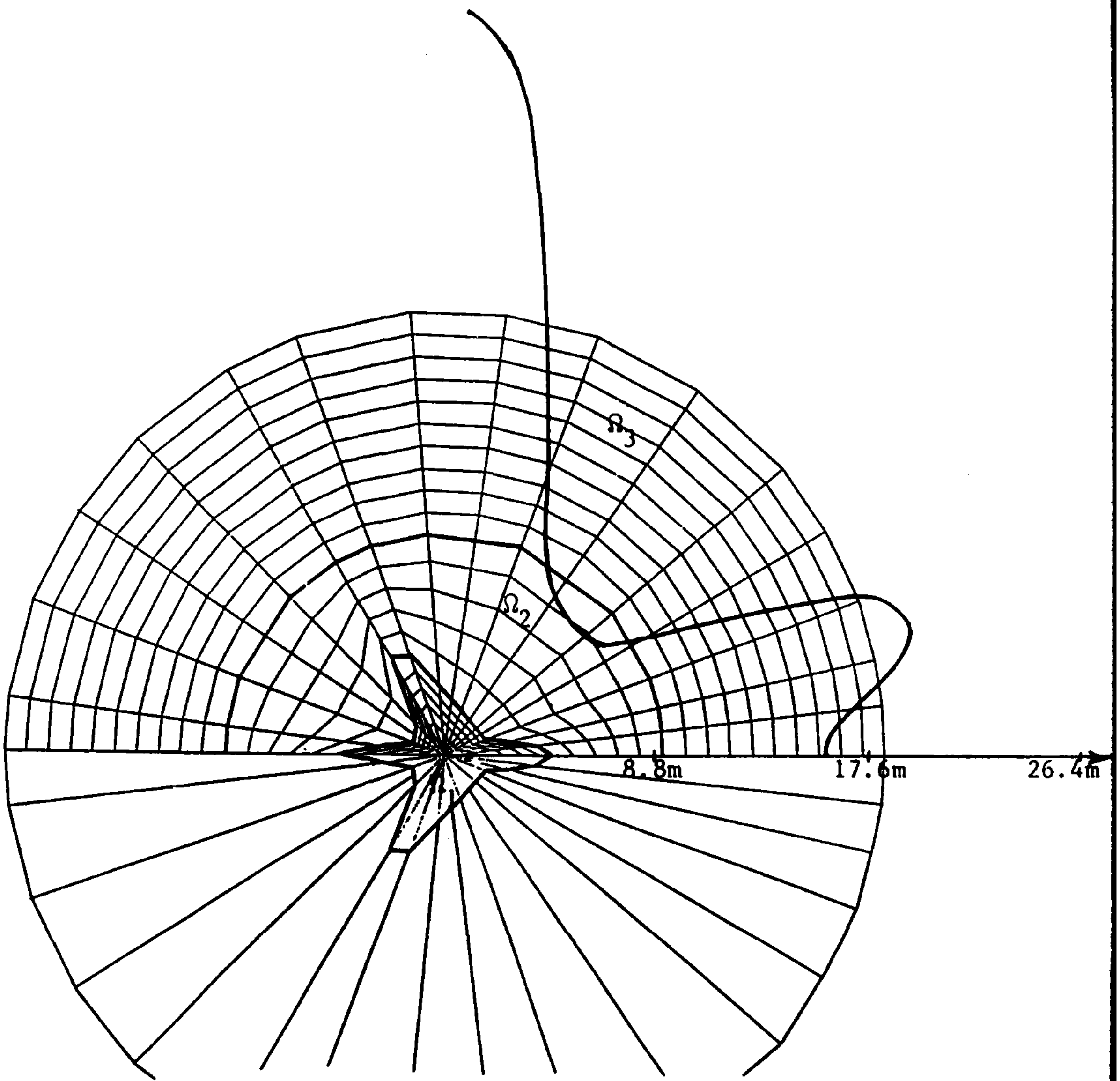


Fig. 9 Radar Back-Scattering Cross Section

To overcome this problem, a multi-grid system should be implemented in Ω_3 to reduce the huge ratio of the grid dimensions. For this, an interpolation scheme must be devised to transmit the information of E-M fields across the boundary separating the finer grid system from the coarser grid system.

There is another computational efficiency problem existing here. For a stable calculation, the smallness of the dimension of the triangular grid zones surrounding the origin of the coordinates makes Δt extremely small, and hence it takes too many Δt to reach a pre-determined time. To overcome this difficulty, one can replace the cluster of small triangular grid zones by a single circular zone and modify the finite difference equations in this neighborhood accordingly.

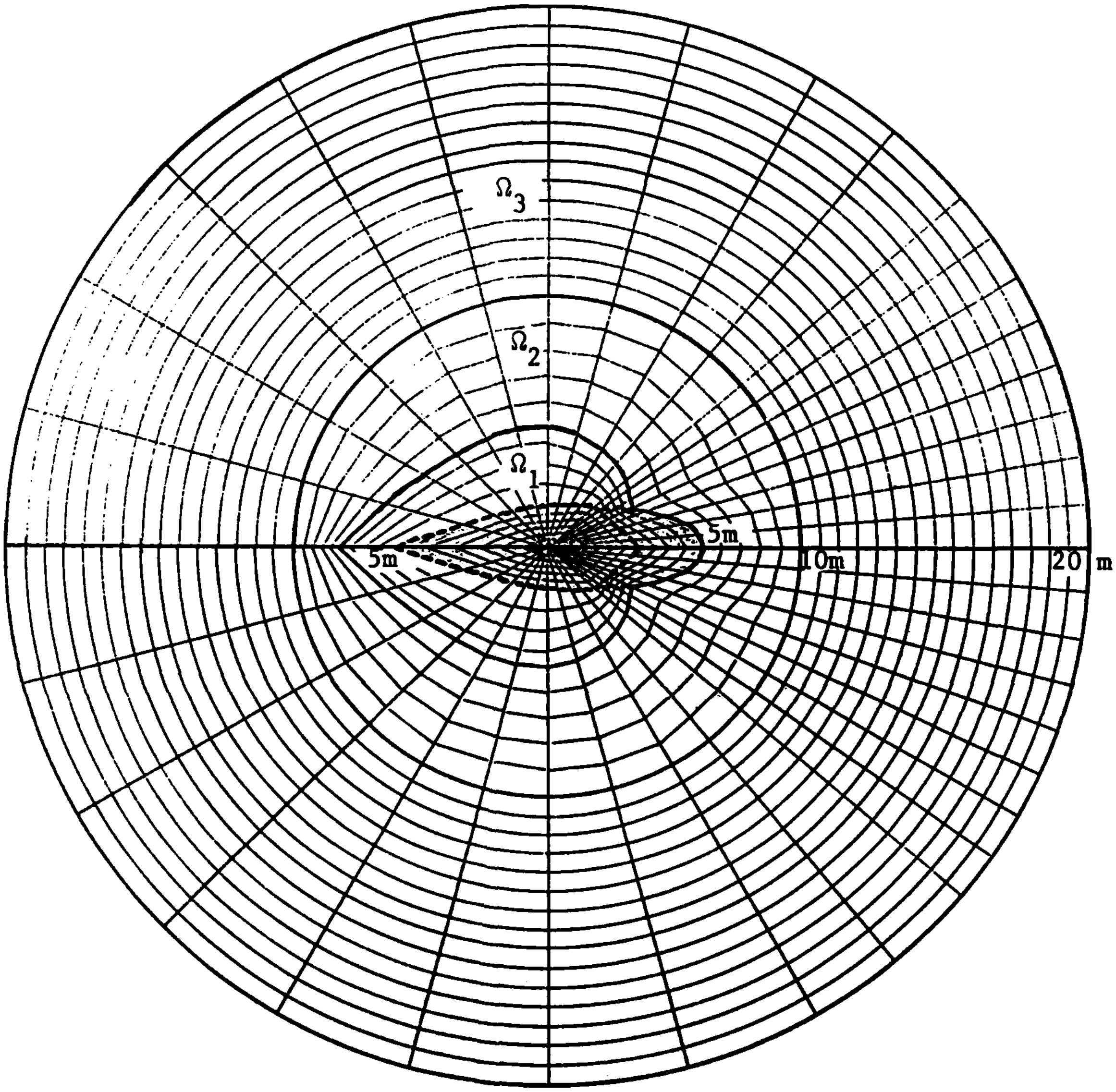


Fig. 10 The shaded area represents the composite airfoil

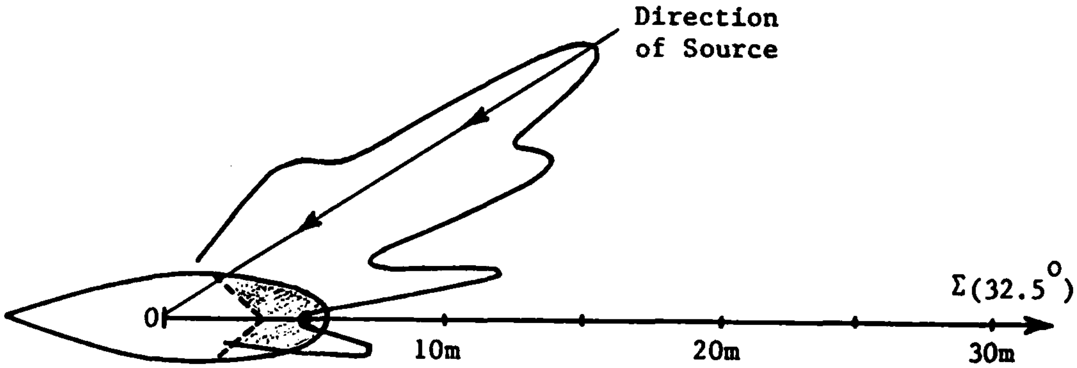


Fig. 11

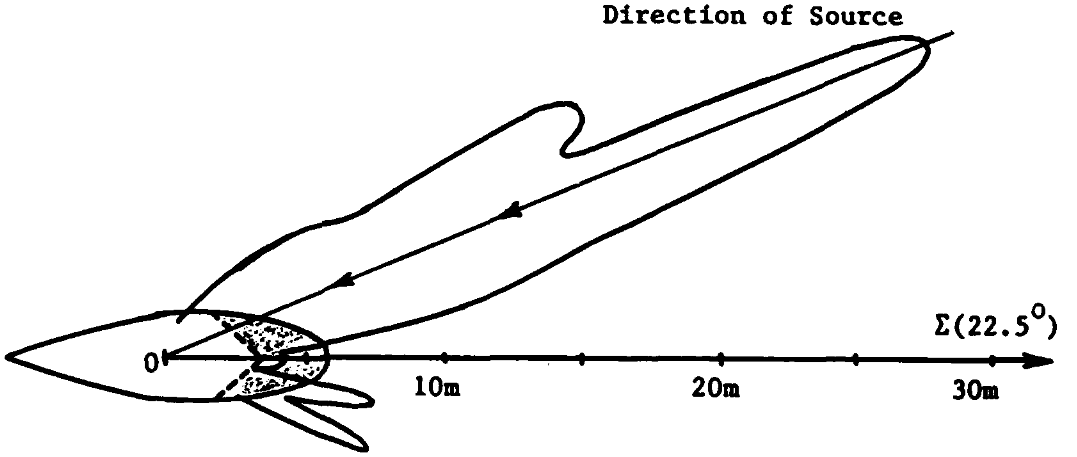


Fig. 12

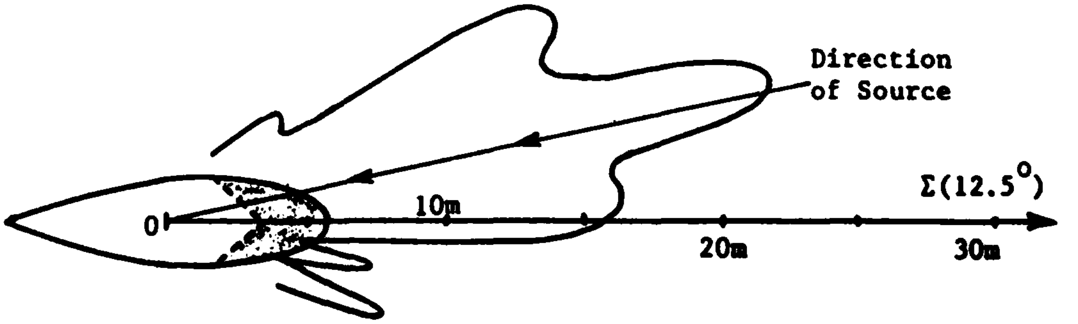


Fig. 13

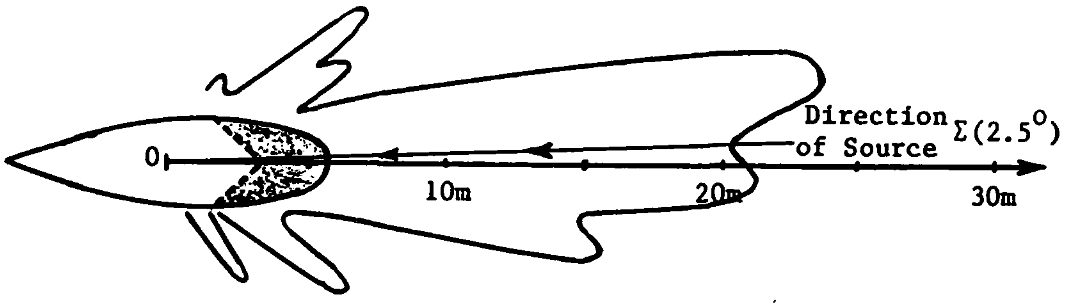


Fig. 14

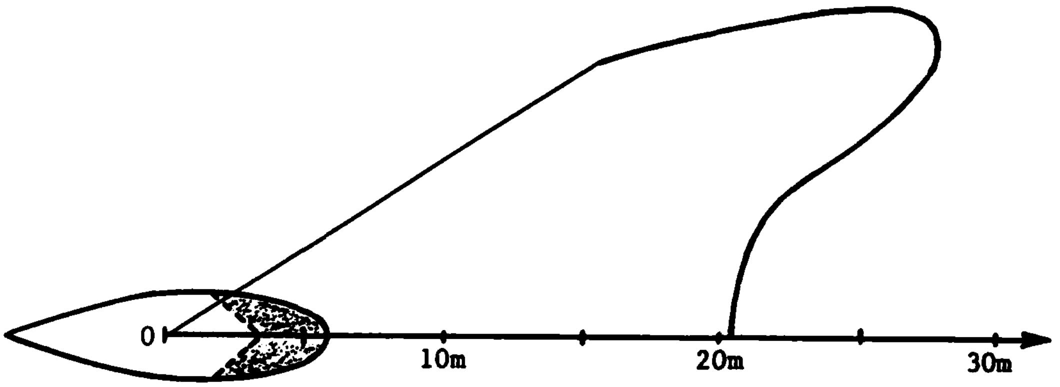


Fig. 15 Radar Back-Scattering Cross Section

REFERENCES

[1] K. S. Yee, "Numerical solution of initial boundary value problems involving Maxwell's equation in isotropic media," IEEE Trans. Ant. Prop., AP-14, pp.302-307, 1966.

[2] A. Taflove and M. E. Brodwin, "Numerical solution of steady-state electromagnetic scattering problems using the time dependent Maxwell's equations," IEEE Trans. Microwave Theo. Tech., MTT-23, pp.623-630, 1975.

[3] R. Holland, "THEDE: A free-field EMP coupling and scattering code," IEEE Trans. Nucl. Sci., NS-24, pp.2416-2421, 1977.

[4] K. S. Kunz and K. M. Lee, "A three-dimensional finite-difference solution of the external response of an aircraft to a complex transient EM environment: Part I - The method and its implementation," IEEE Trans. E-M Comp., EMC-20, pp.328-333, 1978.

[5] Y. M. Chen, "Efficiency of numerical methods for solving Maxwell's equations in space-time domain," SGEMP Note #8, Lawrence Livermore National Lab., Livermore, California, July 1986.

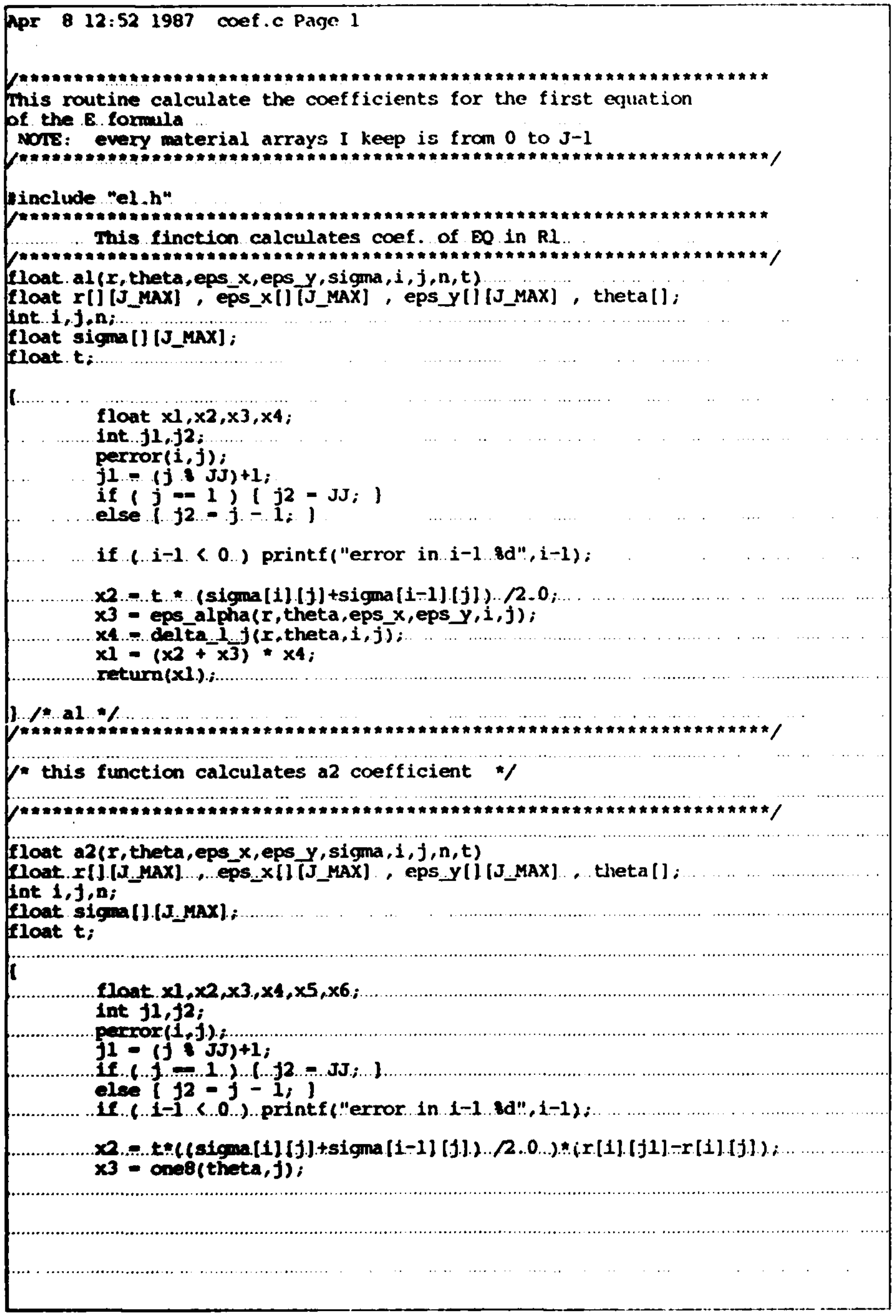
[6] K. K. Mei, A. Cangellaris and D. J. Angelakos, "Conformal time domain finite difference method," Radio Science, 19, pp.1145-1147, 1984.

[7] A. C. Cangellaris, C. C. Lin and K. K. Mei, "Point-matched time domain finite element methods for electromagnetic radiation and scattering," Mem. #UCB/ERL M85/25, Electronic Research Lab., University of California, Berkeley, April, 1985.

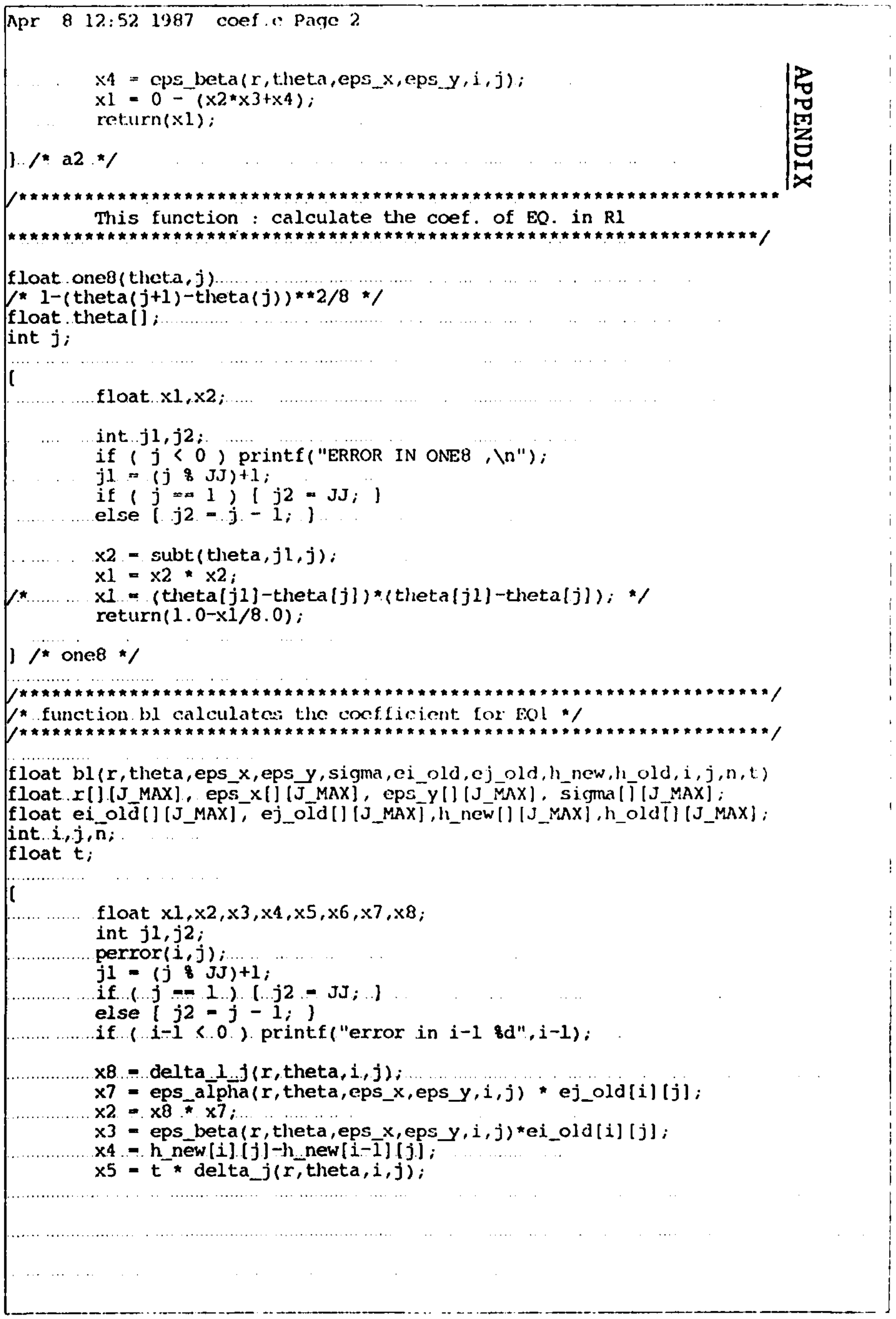
[8] K. S. Yee, "Numerical solution to Maxwell's equations with non-orthogonal grids," SGEMP Note #4, Lawrence Livermore National Lab., Livermore, California, January 1985.

[9] R. W. P. King and T. T. Wu, The Scattering and Diffraction of Waves, Harvard University Press, Cambridge, Mass., 1959.

APPENDIX



```
Apr  8 12:52 1987  coef.c Page 1


/************************************************************
This routine calculate the coefficients for the first equation
of the E formula
 NOTE:  every material arrays I keep is from 0 to J-1
/************************************************************/

#include "el.h"
/************************************************************
        This finction calculates coef. of EQ in R1
/************************************************************/
float a1(r,theta,eps_x,eps_y,sigma,i,j,n,t)
float r[][J_MAX] , eps_x[][J_MAX] , eps_y[][J_MAX] , theta[];
int i,j,n;
float sigma[][J_MAX];
float t;

{
        float x1,x2,x3,x4;
        int j1,j2;
        perror(i,j);
        j1 = (j % JJ)+1;
        if ( j == 1 ) { j2 = JJ; }
        else { j2 = j - 1; }

        if ( i-1 < 0 ) printf("error in i-1 %d",i-1);

        x2 = t * (sigma[i][j]+sigma[i-1][j]) /2.0;
        x3 = eps_alpha(r,theta,eps_x,eps_y,i,j);
        x4 = delta_1_j(r,theta,i,j);
        x1 = (x2 + x3) * x4;
        return(x1);

} /* a1 */
/************************************************************/

/* this function calculates a2 coefficient  */

/************************************************************/

float a2(r,theta,eps_x,eps_y,sigma,i,j,n,t)
float r[][J_MAX] , eps_x[][J_MAX] , eps_y[][J_MAX] , theta[];
int i,j,n;
float sigma[][J_MAX];
float t;

{
        float x1,x2,x3,x4,x5,x6;
        int j1,j2;
        perror(i,j);
        j1 = (j % JJ)+1;
        if ( j == 1 ) { j2 = JJ; }
        else { j2 = j - 1; }
        if ( i-1 < 0 ) printf("error in i-1 %d",i-1);

        x2 = t*((sigma[i][j]+sigma[i-1][j]) /2.0 )*(r[i][j1]-r[i][j]);
        x3 = one8(theta,j);
```



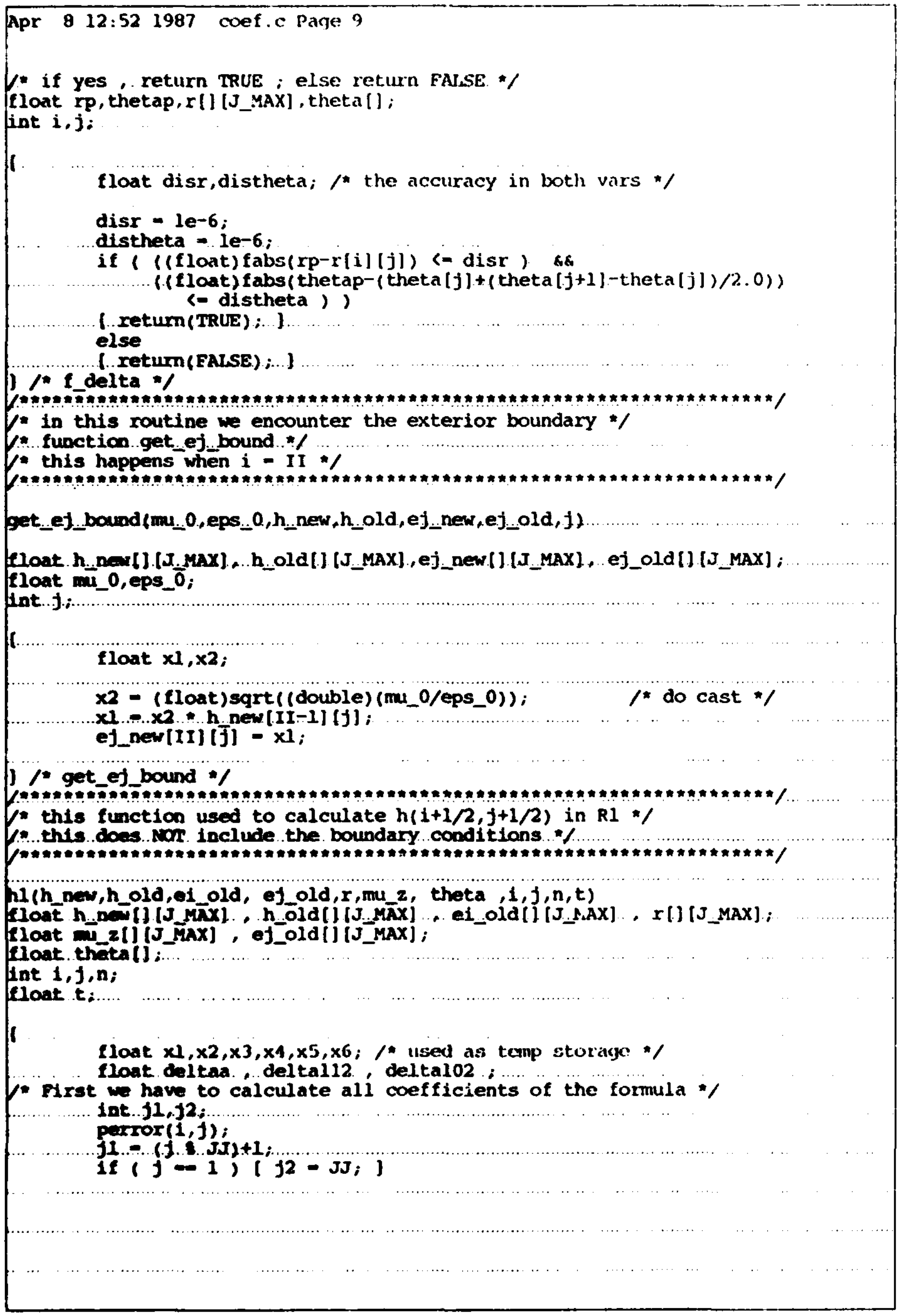
```
Apr  8 12:52 1987  coef.c Page 2

        x4 = eps_beta(r,theta,eps_x,eps_y,i,j);
        x1 = 0 - (x2*x3+x4);
        return(x1);

} /* a2 */

/************************************************************
        This function : calculate the coef. of EQ. in R1
************************************************************/

float one8(theta,j)
/* 1-(theta(j+1)-theta(j))**2/8 */
float theta[];
int j;

{
        float x1,x2;

        int j1,j2;
        if ( j < 0 ) printf("ERROR IN ONE8 ,\n");
        j1 = (j % JJ)+1;
        if ( j == 1 ) { j2 = JJ; }
        else { j2 = j - 1; }

        x2 = subt(theta,j1,j);
        x1 = x2 * x2;
/*      x1 = (theta[j1]-theta[j])*(theta[j1]-theta[j]); */
        return(1.0-x1/8.0);

} /* one8 */

/************************************************************/
/* function b1 calculates the coefficient for EQ1 */
/************************************************************/

float b1(r,theta,eps_x,eps_y,sigma,ei_old,ej_old,h_new,h_old,i,j,n,t)
float r[][J_MAX], eps_x[][J_MAX], eps_y[][J_MAX], sigma[][J_MAX];
float ei_old[][J_MAX], ej_old[][J_MAX],h_new[][J_MAX],h_old[][J_MAX];
int i,j,n;
float t;

{
        float x1,x2,x3,x4,x5,x6,x7,x8;
        int j1,j2;
        perror(i,j);
        j1 = (j % JJ)+1;
        if ( j == 1 ) { j2 = JJ; }
        else { j2 = j - 1; }
        if ( i-1 < 0 ) printf("error in i-1 %d",i-1);

        x8 = delta_1_j(r,theta,i,j);
        x7 = eps_alpha(r,theta,eps_x,eps_y,i,j) * ej_old[i][j];
        x2 = x8 * x7;
        x3 = eps_beta(r,theta,eps_x,eps_y,i,j)*ei_old[i][j];
        x4 = h_new[i][j]-h_new[i-1][j];
        x5 = t * delta_j(r,theta,i,j);
        x6 = upperr(r,i,j)-upperr(r,i-1,j);
        x1 = x2-x3-x5*x4/x6;
        return(x1);

} /* b1 */
/*******************************************************************/
/* function a3 to calculate coefficient for EQ2 in R1 */
/*******************************************************************/

float a3(r,theta,eps_x,eps_y,sigma,i,j,n,t)
float r[][J_MAX] , eps_x[][J_MAX] , eps_y[][J_MAX] , theta[];
int i,j,n;
float sigma[][J_MAX];
float t;

{
        float x1,x2,x3,x4,x5,x6,x7;
        int j1,j2;
        perror(i,j);
        j1 = (j % JJ)+1;
        if ( j == 1 ) { j2 = JJ; }
        else { j2 = j - 1; }

        x2 = t*(sigma[i][j]+sigma[i][j2])/2.0;
        x3 = delta_l_i(r,theta,i,j);
        x4 = eps_lu(r,theta,eps_x,eps_y,i,j);
        x5 = upperr(r,i,j)*one8(theta,j);
        x6 = lowerr(r,i,j)*one8(theta,j2);
        x7 = (x2*(x5-x6)+x4)*x3;
        x1 = 0-x7;
        return(x1);

} /* a3 */

/*******************************************************************/
/* function a4 to calculate coefficient for EQ2 */
/*******************************************************************/

float a4(r,theta,eps_x,eps_y,sigma,i,j,n,t)
float r[][J_MAX] , eps_x[][J_MAX] , eps_y[][J_MAX] , theta[];
int i,j,n;
float sigma[][J_MAX];
float t;

{
        float x1,x2,x3,x4,x5;
        int j1,j2;
        perror(i,j);
        j1 = (j % JJ)+1;
        if ( j == 1 ) { j2 = JJ; }
        else { j2 = j - 1; }

        x2 = t*(sigma[i][j]+sigma[i][j2])/2.0;
        x3 = delta_l_i(r,theta,i,j);
        x4 = eps_lamda(r,theta,eps_x,eps_y,i,j);
        x1 = x2*x3*x3+x4;
```

```
        return(x1);

} /* a4 */
/*******************************************************************/
/* calculate coefficient for EQ2 */
/*******************************************************************/

float b2(r,theta,eps_x,eps_y,sigma,ei_old,ej_old,h_new,h_old,i,j,n,t)
float r[][J_MAX], eps_x[][J_MAX], eps_y[][J_MAX], sigma[][J_MAX];
float ei_old[][J_MAX], ej_old[][J_MAX],h_new[][J_MAX],h_old[][J_MAX];
int i,j,n;
float t;

{
        float x1,x2,x3,x4,x5,x6,x7,x8;
        int j1,j2;
        perror(i,j);
        j1 = (j % JJ)+1;
        if ( j == 1 ) { j2 = JJ; }
        else { j2 = j - 1; }

        x2 = delta_l_i(r,theta,i,j);
        x3 = eps_lu(r,theta,eps_x,eps_y,i,j);
        x4 = x2*x3*ej_old[i][j];
        x5 = eps_lamda(r,theta,eps_x,eps_y,i,j);
        x6 = x5*ei_old[i][j];
        x7 = h_new[i][j]-h_new[i][j2];
        x8 = t*delta_i(r,theta,i,j);
        x1 = x6 + x8*x7 - x4;
        return(x1);

} /* b2 */
/*******************************************************************/
/* function a5 to calculate coefficient for EQ1 in R2 */
/*******************************************************************/

float a5(r,theta,eps_0,mu_0,sigma_0,i,j,n,t)
float r[][J_MAX], theta[];
int i,j,n;
float eps_0,mu_0,sigma_0,t;

{
        float x1,x2,x3;
        int j1,j2;
        perror(i,j);
        j1 = (j % JJ)+1;
        if ( j == 1 ) { j2 = JJ; }
        else { j2 = j - 1; }

        x2 = t*sigma_0+eps_0;
        x3 = delta_l_j(r,theta,i,j);
        x1 = x2*x3;
        return(x1);

} /* a5 */
```

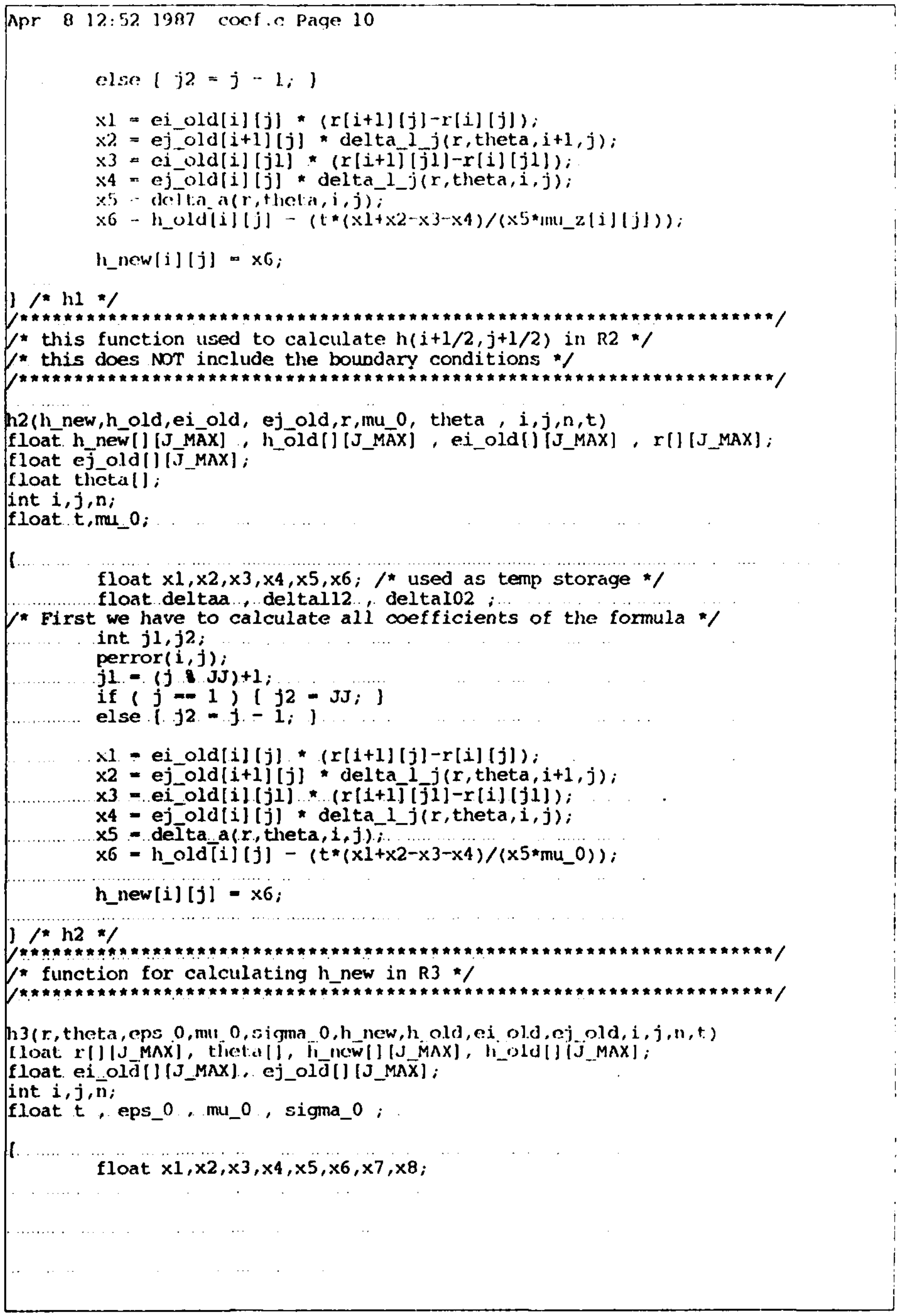
```
/****************************************************************/
/* calculates a6 coefficient for EQ1 in R2 */
/******************************************************************/

float a6(r,theta,eps_0,mu_0,sigma_0,i,j,n,t)
float r[][J_MAX], theta[];
int i,j,n;
float eps_0,mu_0,sigma_0,t;

{
        float x1,x2,x3,x4;
        int j1,j2;
        perror(i,j);
        j1 = (j % JJ)+1;
        if ( j == 1 ) { j2 = JJ; }
        else { j2 = j - 1; }

        x2 = t*sigma_0+eps_0;
        x3 = one8(theta,j);
        x4 = r[i][j1]-r[i][j];
        x1 = 0-x2*x3*x4;
        return(x1);

} /* a6 */

/******************************************************************/
/* function b3 for coefficient for EQ2 in R2 */
/******************************************************************/

float b3(r,theta,eps_0,sigma_0,mu_0,ei_old,ej_old,h_new,h_old,i,j,n,t)
float r[][J_MAX], theta[];
float ei_old[][J_MAX], ej_old[][J_MAX],h_new[][J_MAX],h_old[][J_MAX];
int i,j,n;
float t,eps_0,sigma_0,mu_0;

{
        float x1,x2,x3,x4,x5,x6,x7,x8,x9;
        int j1,j2;
        perror(i,j);
        j1 = (j % JJ)+1;
        if ( j == 1 ) { j2 = JJ; }
        else { j2 = j - 1; }

        if ( i-1 < 0 ) printf("error in i-1 %d",i-1);

        x2 = delta_l_j(r,theta,i,j)*eps_0*ej_old[i][j];
        x3 = one8(theta,j);
        x4 = eps_0*(r[i][j1]-r[i][j]) * ei_old[i][j];
        x5 = x3*x4;
        x6 = t*delta_j(r,theta,i,j);
        x7 = h_new[i][j]-h_new[i 1][j];
        x8 = upperr(r,i,j)-upperr(r,i-1,j); /* this will r,0,j */
        x9 = x6*x7/x8;
        x1 = x2-x5-x9;
        return(x1);
```

```
} /* b3 */

/******************************************************************/
/* function a7 for calculating coefficient in EQ2 in R2 */
/******************************************************************/
float a7(r,theta,eps_0,mu_0,sigma_0,i,j,n,t)
float r[][J_MAX], theta[];
int i,j,n;
float eps_0,mu_0,sigma_0,t;

{
        float x1,x2,x3,x4,x5,x6;
        int j1,j2;
        perror(i,j);
        j1 = (j % JJ)+1;
        if ( j == 1 ) { j2 = JJ; }
        else { j2 = j - 1; }

        x2 = t * sigma_0 + eps_0;
        x3 = delta_l_i(r,theta,i,j);
        x4 = one8(theta,j)*upperr(r,i,j);
        x5 = one8(theta,j2)*upperr(r,i,j2);
        x6 = x2 * x3 * (x4-x5);
        x1 = 0 - x6;
        return(x1);

} /* a7 */

/******************************************************************/
/* function a8 for calculating coefficient for EQ2 in R2 */
/******************************************************************/
float a8(r,theta,eps_0,mu_0,sigma_0,i,j,n,t)
float r[][J_MAX], theta[];
int i,j,n;
float eps_0,mu_0,sigma_0,t;

{
        float x1,x2,x3,x4,x5,x6;
        int j1,j2;
        perror(i,j);
        j1 = (j % JJ)+1;
        if ( j == 1 ) { j2 = JJ; }
        else { j2 = j - 1; }

        x2 = delta_l_i(r,theta,i,j);
        x3 = t * sigma_0+eps_0;
        x1 = x2 * x2 * x3;
        return(x1);

} /* a8 */

/******************************************************************/
/* function b4 to calculate coefficient for EQ2 in R2 */
/******************************************************************/

float b4(r,theta,eps_0,sigma_0,mu_0,ei_old,ej_old,h_new,h_old,i,j,n,t)
float r[][J_MAX], theta[];
float ei_old[][J_MAX], ej_old[][J_MAX],h_new[][J_MAX],h_old[][J_MAX];
int i,j,n;
float t,eps_0,sigma_0,mu_0;

{
        float x1,x2,x3,x4,x5,x6,x7,x8,x9,x10,x11,x12;
        int j1,j2;
        perror(i,j);
        j1 = (j % JJ)+1;
        if ( j == 1 ) { j2 = JJ; }
        else { j2 = j - 1; }

        x2 = delta_l_i(r,theta,i,j) * eps_0 * ej_old[i][j];
        x3 = one8(theta,j) * upperr(r,i,j);
        x4 = one8(theta,j2) * lowerr(r,i,j);
        x5 = x2 * (x3-x4);
        x6 = eps_0 * ei_old[i][j];
        x7 = delta_l_i(r,theta,i,j);
        x8 = x7*x7*x6;
        x9 = h_new[i][j]-h_new[i][j2];
        x10 = t * delta_i(r,theta,i,j) * x9;
        x1 = x10 + x8 - x5;
        return(x1);

} /* b4 */
/*****************************************************************/
/* in R3 , function are simple and elegant */
/*****************************************************************/

get_ej(r,theta,h_new,h_old,ej_new,ej_old,eps_0,sigma_0,mu_0,
        i,j,n,t,rp,thetap)
float r[][J_MAX],theta[],h_new[][J_MAX],h_old[][J_MAX];
float ej_new[][J_MAX],ej_old[][J_MAX];
int i,j,n;
float eps_0,sigma_0,mu_0,t,rp,thetap;

{
        float x1,x2,x3,x4,x5,x6,x7,x8,x9,x10;
        int j1,j2;
        perror(i,j);
        j1 = (j % JJ)+1;
        if ( j == 1 ) { j2 = JJ; }
        else { j2 = j - 1; }

        if ( i-1 < 0 ) printf("error in i-1 %d",i-1);
        x2 = h_new[i][j] - h_new[i-1][j];
        x3 = 2*t/(r[i+1][j]-r[i-1][j]);
        x4 = eps_0*ej_old[i][j];
        x5 = t*sigma_0+eps_0;
        x1 = (x4-x3*x2) / x5;
        if(f_delta(rp,thetap,r,theta,i,j) ) {
/*              x6 = (theta[j1]+theta[j]) / 2.0; */
                x6 = addt(theta,j1,j) / 2.0;
                x7 = (float)sin((double)x6);
                x8 = (float)cos((double)x6);
```

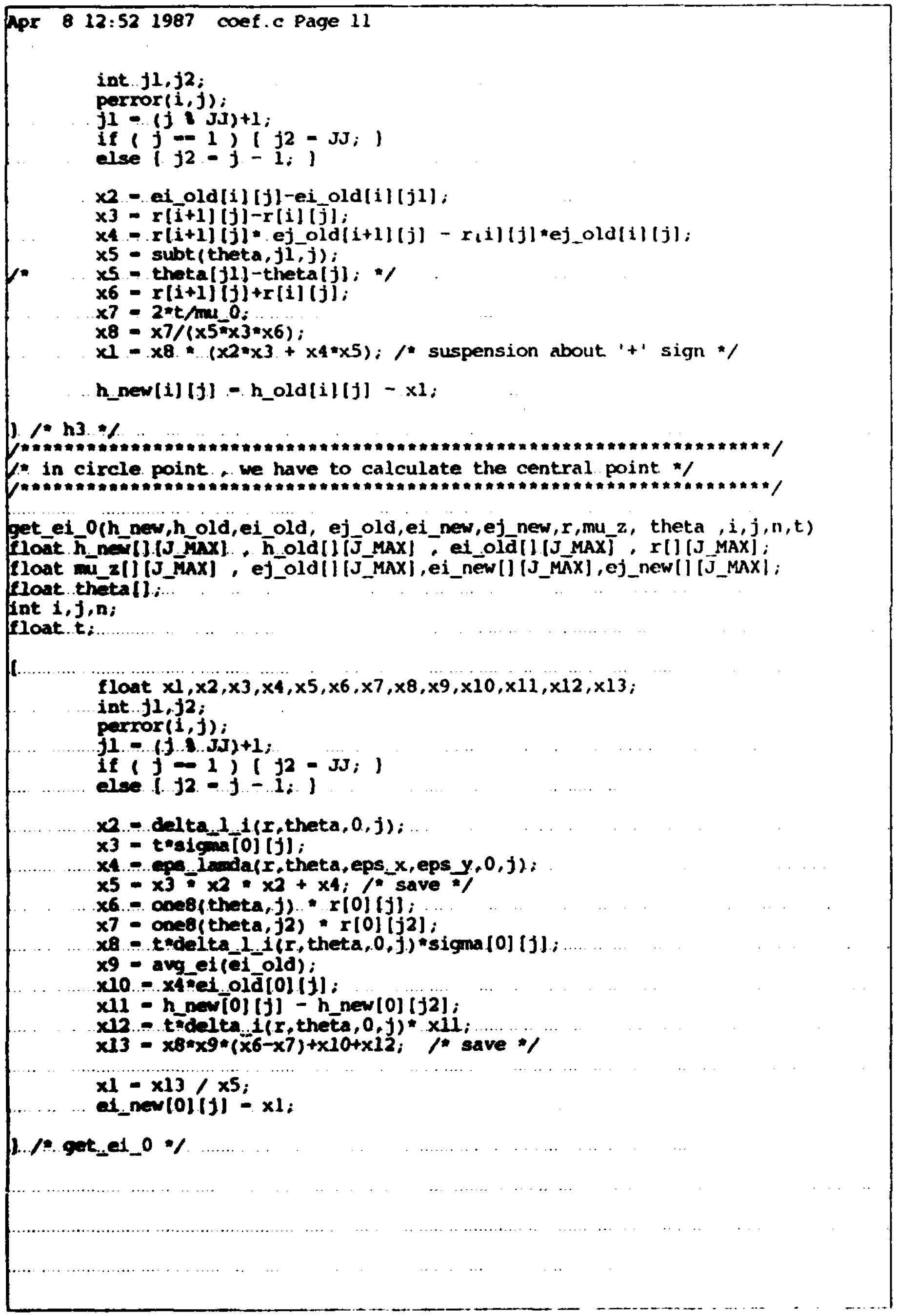
```
                x9 = funcjx(j,n,t);
                x10 = funcjy(j,n,t);
                x1 = x1 - t*(x10*x8 - x9*x7) / x5;
        } /* endif */

        ej_new[i][j] = x1;
} /* get_ej */

/*****************************************************************/
/* get_ei function : calculate ei_field in R3 */
/******************************************************************/
get_ei(r,theta,h_new,h_old,ei_new,ei_old,eps_0,sigma_0,mu_0,
        i,j,n,t,rp,thetap)
float r[][J_MAX],theta[],h_new[][J_MAX],h_old[][J_MAX];
float ei_new[][J_MAX],ei_old[][J_MAX];
int i,j,n;
float eps_0,sigma_0,mu_0,t,rp,thetap;

{
        float x1,x2,x3,x4,x5,x6,x7,x8,x9,x10;
        int j1,j2;
        perror(i,j);
        j1 = (j % JJ)+1;
        if ( j == 1 ) { j2 = JJ; }
        else { j2 = j - 1; }

        x2 = h_new[i][j] - h_new[i][j2];
        x3 = subt(theta,j1,j2);
/*      x3 = theta[j1]-theta[j2]; */
        x4 = r[i+1][j] + r[i][j];
        x5 = eps_0 * ei_old[i][j];
        x6 = t*sigma_0 + eps_0;
        x7 = 4*t*x2/(x4*x3);
        x1 = (x5 + x7) / x6;

        if ( f_delta(rp,thetap,r,theta,i,j)) {
/*              x2 = (theta[j1]+theta[j])/2.0; */
/*              x3 = (theta[j]+theta[j2])/2.0; */
                x2 = addt(theta,j1,j) / 2.0;
                x3 = addt(theta,j,j2) / 2.0;
                x4 = funcjx(j,n,t);
                x5 = funcjy(j,n,t);
        x7 = t*(x5*((float)cos((double)x2)-(float)cos((double)x3))-
                x4*((float)sin((double)x2)-(float)sin((double)x3)));
        x1 = x1 - x7/x6;
        } /* endif */

        ei_new[i][j] = x1;

} /* get_ei */

/*******************************************************************/
/* this is a boolean function to test if the grid point is in the same
   location as a given radar point
******************************************************************/
int f_delta(rp,thetap,r,theta,i,j)
```



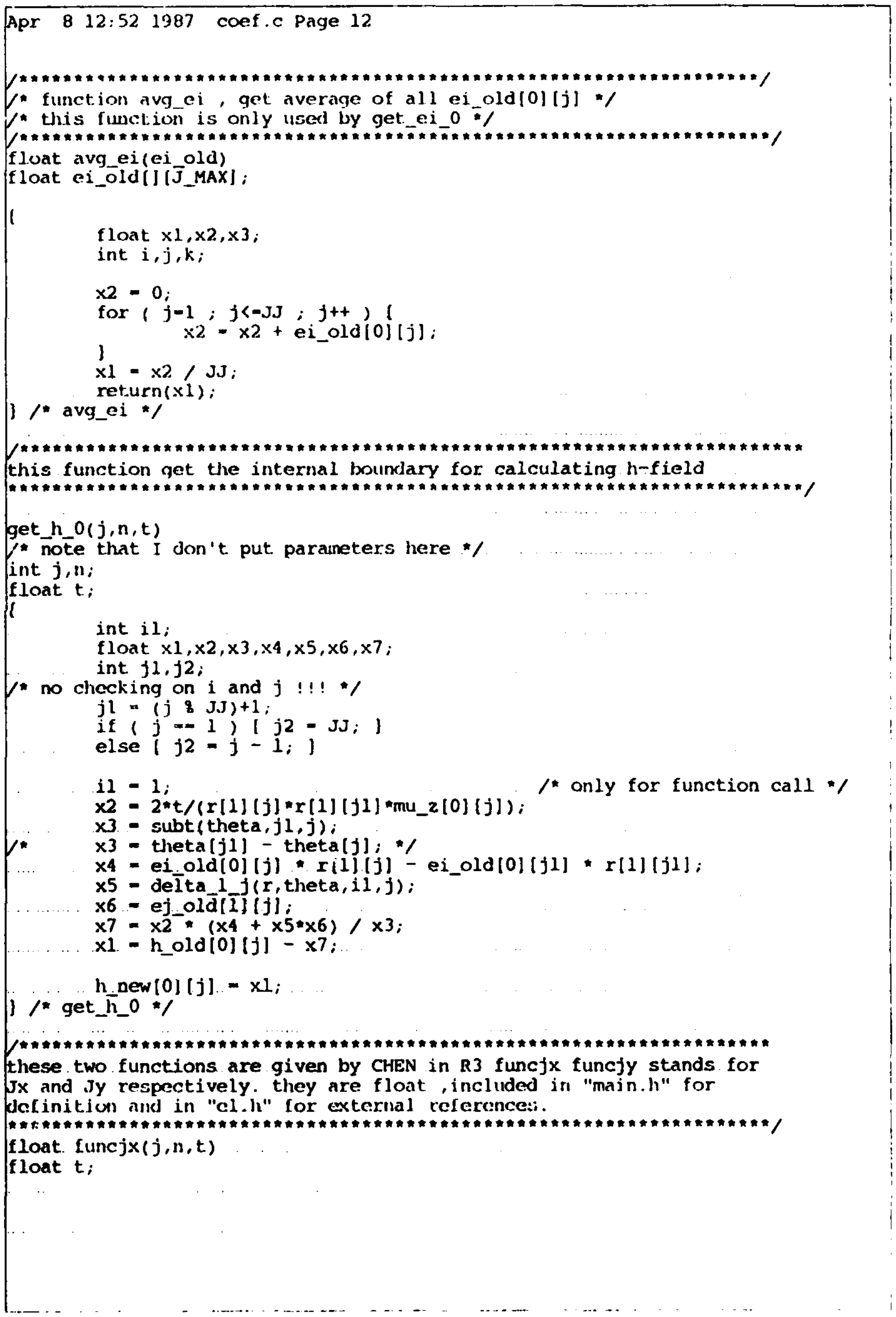
```
/* if yes , return TRUE ; else return FALSE */
float rp,thetap,r[][J_MAX],theta[];
int i,j;

{
        float disr,distheta; /* the accuracy in both vars */

        disr = le-6;
        distheta = le-6;
        if ( ((float)fabs(rp-r[i][j]) <= disr )  &&
             ((float)fabs(thetap-(theta[j]+(theta[j+1]-theta[j])/2.0))
                <= distheta ) )
        { return(TRUE); }
        else
        { return(FALSE); }
} /* f_delta */
/*****************************************************************/
/* in this routine we encounter the exterior boundary */
/* function get_ej_bound */
/* this happens when i = II */
/*****************************************************************/

get_ej_bound(mu_0,eps_0,h_new,h_old,ej_new,ej_old,j)

float h_new[][J_MAX], h_old[][J_MAX],ej_new[][J_MAX], ej_old[][J_MAX];
float mu_0,eps_0;
int j;

{
        float x1,x2;

        x2 = (float)sqrt((double)(mu_0/eps_0));          /* do cast */
        x1 = x2 * h_new[II-1][j];
        ej_new[II][j] = x1;

} /* get_ej_bound */
/*****************************************************************/
/* this function used to calculate h(i+1/2,j+1/2) in R1 */
/* this does NOT include the boundary conditions */
/*****************************************************************/

h1(h_new,h_old,ei_old, ej_old,r,mu_z, theta ,i,j,n,t)
float h_new[][J_MAX] , h_old[][J_MAX] , ei_old[][J_MAX] , r[][J_MAX];
float mu_z[][J_MAX] , ej_old[][J_MAX];
float theta[];
int i,j,n;
float t;

{
        float x1,x2,x3,x4,x5,x6; /* used as temp storage */
        float deltaa , deltal12 , deltal02 ;
/* First we have to calculate all coefficients of the formula */
        int j1,j2;
        perror(i,j);
        j1 = (j % JJ)+1;
        if ( j == 1 ) { j2 = JJ; }
```



```
        else { j2 = j - 1; }

        x1 = ei_old[i][j] * (r[i+1][j]-r[i][j]);
        x2 = ej_old[i+1][j] * delta_l_j(r,theta,i+1,j);
        x3 = ei_old[i][j1] * (r[i+1][j1]-r[i][j1]);
        x4 = ej_old[i][j] * delta_l_j(r,theta,i,j);
        x5 = delta_a(r,theta,i,j);
        x6 = h_old[i][j] - (t*(x1+x2-x3-x4)/(x5*mu_z[i][j]));

        h_new[i][j] = x6;

} /* h1 */
/*****************************************************************/
/* this function used to calculate h(i+1/2,j+1/2) in R2 */
/* this does NOT include the boundary conditions */
/*****************************************************************/

h2(h_new,h_old,ei_old, ej_old,r,mu_0, theta , i,j,n,t)
float h_new[][J_MAX] , h_old[][J_MAX] , ei_old[][J_MAX] , r[][J_MAX];
float ej_old[][J_MAX];
float theta[];
int i,j,n;
float t,mu_0;

{
        float x1,x2,x3,x4,x5,x6; /* used as temp storage */
        float deltaa , deltal12 , deltal02 ;
/* First we have to calculate all coefficients of the formula */
        int j1,j2;
        perror(i,j);
        j1 = (j % JJ)+1;
        if ( j == 1 ) { j2 = JJ; }
        else { j2 = j - 1; }

        x1 = ei_old[i][j] * (r[i+1][j]-r[i][j]);
        x2 = ej_old[i+1][j] * delta_l_j(r,theta,i+1,j);
        x3 = ei_old[i][j1] * (r[i+1][j1]-r[i][j1]);
        x4 = ej_old[i][j] * delta_l_j(r,theta,i,j);
        x5 = delta_a(r,theta,i,j);
        x6 = h_old[i][j] - (t*(x1+x2-x3-x4)/(x5*mu_0));

        h_new[i][j] = x6;

} /* h2 */
/*****************************************************************/
/* function for calculating h_new in R3 */
/*****************************************************************/

h3(r,theta,eps_0,mu_0,sigma_0,h_new,h_old,ei_old,ej_old,i,j,n,t)
float r[][J_MAX], theta[], h_new[][J_MAX], h_old[][J_MAX];
float ei_old[][J_MAX], ej_old[][J_MAX];
int i,j,n;
float t , eps_0 , mu_0 , sigma_0 ;

{
        float x1,x2,x3,x4,x5,x6,x7,x8;
```



```
Apr  8 12:52 1987  coef.c Page 11

        int j1,j2;
        perror(i,j);
        j1 = (j % JJ)+1;
        if ( j == 1 ) { j2 = JJ; }
        else { j2 = j - 1; }

        x2 = ei_old[i][j]-ei_old[i][j1];
        x3 = r[i+1][j]-r[i][j];
        x4 = r[i+1][j]* ej_old[i+1][j] - r[i][j]*ej_old[i][j];
        x5 = subt(theta,j1,j);
/*      x5 = theta[j1]-theta[j]; */
        x6 = r[i+1][j]+r[i][j];
        x7 = 2*t/mu_0;
        x8 = x7/(x5*x3*x6);
        x1 = x8 * (x2*x3 + x4*x5); /* suspension about '+' sign */

        h_new[i][j] = h_old[i][j] - x1;

} /* h3 */
/********************************************************************/
/* in circle point , we have to calculate the central point */
/********************************************************************/

get_ei_0(h_new,h_old,ei_old, ej_old,ei_new,ej_new,r,mu_z, theta ,i,j,n,t)
float h_new[][J_MAX] , h_old[][J_MAX] , ei_old[][J_MAX] , r[][J_MAX];
float mu_z[][J_MAX] , ej_old[][J_MAX],ei_new[][J_MAX],ej_new[][J_MAX];
float theta[];
int i,j,n;
float t;

{
        float x1,x2,x3,x4,x5,x6,x7,x8,x9,x10,x11,x12,x13;
        int j1,j2;
        perror(i,j);
        j1 = (j % JJ)+1;
        if ( j == 1 ) { j2 = JJ; }
        else { j2 = j - 1; }

        x2 = delta_l_i(r,theta,0,j);
        x3 = t*sigma[0][j];
        x4 = eps_lamda(r,theta,eps_x,eps_y,0,j);
        x5 = x3 * x2 * x2 + x4; /* save */
        x6 = one8(theta,j) * r[0][j];
        x7 = one8(theta,j2) * r[0][j2];
        x8 = t*delta_l_i(r,theta,0,j)*sigma[0][j];
        x9 = avg_ei(ei_old);
        x10 = x4*ei_old[0][j];
        x11 = h_new[0][j] - h_new[0][j2];
        x12 = t*delta_i(r,theta,0,j)* x11;
        x13 = x8*x9*(x6-x7)+x10+x12;  /* save */

        x1 = x13 / x5;
        ei_new[0][j] = x1;

} /* get_ei_0 */
```



```
Apr  8 12:52 1987  coef.c Page 12

/********************************************************************/
/* function avg_ei , get average of all ei_old[0][j] */
/* this function is only used by get_ei_0 */
/********************************************************************/
float avg_ei(ei_old)
float ei_old[][J_MAX];

{
        float x1,x2,x3;
        int i,j,k;

        x2 = 0;
        for ( j=1 ; j<=JJ ; j++ ) {
                x2 = x2 + ei_old[0][j];
        }
        x1 = x2 / JJ;
        return(x1);
} /* avg_ei */

/********************************************************************
this function get the internal boundary for calculating h-field
*********************************************************************/

get_h_0(j,n,t)
/* note that I don't put parameters here */
int j,n;
float t;
{
        int i1;
        float x1,x2,x3,x4,x5,x6,x7;
        int j1,j2;
/* no checking on i and j !!! */
        j1 = (j % JJ)+1;
        if ( j == 1 ) { j2 = JJ; }
        else { j2 = j - 1; }

        i1 = 1;                                 /* only for function call */
        x2 = 2*t/(r[1][j]*r[1][j1]*mu_z[0][j]);
        x3 = subt(theta,j1,j);
/*      x3 = theta[j1] - theta[j]; */
        x4 = ei_old[0][j] * r[1][j] - ei_old[0][j1] * r[1][j1];
        x5 = delta_l_j(r,theta,i1,j);
        x6 = ej_old[1][j];
        x7 = x2 * (x4 + x5*x6) / x3;
        x1 = h_old[0][j] - x7;

        h_new[0][j] = x1;
} /* get_h_0 */

/********************************************************************
these two functions are given by CHEN in R3 funcjx funcjy stands for
Jx and Jy respectively. they are float ,included in "main.h" for
definition and in "cl.h" for external references.
********************************************************************/
float funcjx(j,n,t)
float t;
```

```
Apr  8 12:52 1987  coef.c Page 13


int j,n;

/* note that in this function I have to use  theta as global vars */

{
        float x1,x2,x3,x4,x5;

        x2 = (n*t-5.0*t) * (n*t-5.0*t);
        x3 = 5*t*t;
        x4 = exp(0 - x2/x3);
        x5 = (float)sin((double)(theta[j]+(theta[j+1]-theta[j])/2.0 - 0.05236));
        x1 = x4*x5;

        return(x1);
} /* funcjx */

/***************************************************************
        This function : calculate source wave
***************************************************************/

float funcjy(j,n,t)
float t;
int j,n;

/* note that in this function I have to use theta as global vars */

{
        float x1,x2,x3,x4,x5;

        x2 = (n*t-5.0*t) * (n*t-5.0*t);
        x3 = 5*t*t;
        x4 = exp(0 - x2/x3);
        x5 = (float)cos((double)(theta[j]+(theta[j+1]-theta[j])/2.0 - 0.05236));
        x1 = x4*x5;

        return(x1);
} /* funcjy */
```

```
Apr  1 13:14 1987  delta.c Page 1


/***************************************************************/
/* function for getting delta_a */
/* delta_a(i+1/2,j+1/2) */
/***************************************************************/
#include "util.h"

float  delta_a(r,theta,i,j)
float r[][J_MAX] , theta[];
int i,j;

{
        float x2,x3;
        float x1; /* delta_a(i+1/2,j+1/2) */

        int j1,j2;
        perror(i,j);
        j1 = (j % JJ)+1;
        if ( j == 1 ) { j2 = JJ; }
        else { j2 = j - 1; }

        x2 = subt(theta,j1,j) / 2.0;
/*      x2 = (float)fabs(theta[j1]-theta[j]) / 2.0; */
        x3 = (r[i+1][j] - r[i][j]) * r[i][j1] + (r[i+1][j1]-
             r[i][j1]) * r[i+1][j] ;
        x1 = x2 * x3 ;
        return(x1);
} /* delta_a */
/***************************************************************/
/* function delta_i */
/* delta(i+1/2,j) */
/***************************************************************/


float delta_i(r,theta,i,j)
float r[][J_MAX], theta[];
int i,j;

{
        float x2,x3,x4,x5;
        float x1; /* = delta_i(i+1/2,j) */
        int j1,j2;
        perror(i,j);
        j1 = (j % JJ)+1;
        if ( j == 1 ) { j2 = JJ; }
        else { j2 = j - 1; }

        x2 = upperr(r,i,j);
        x3 = lowerr(r,i,j);

/*      x1 = ((float)fabs(theta[j1]-theta[j])*x2 +
             (float)fabs(theta[j]-theta[j2])*x3)/2; */
        x4 = subt(theta,j1,j); x5 = subt(theta,j,j2);
        x1 = (x4 * x2 + x5 * x3 )/2.0;
        return(x1);

} /* delta_i */
```

```
Apr  1 13:14 1987  delta.c Page 2


/*************************************************************/
/* function delta_j */
/*************************************************************/


float  delta_j(r,theta,i,j)
float r[][J_MAX], theta[];
int i,j;

{
        float x1; /* delta(i,j+1/2) */
        int j1,j2;
        perror(i,j);
        j1 = (j % JJ)+1;
        if ( j == 1 ) { j2 = JJ; }
        else { j2 = j - 1; }

        x1 = (r[i][j1]+r[i][j]) * subt(theta,j1,j) / 2.0;
        return(x1);
} /* delta_j */
/*************************************************************/
/* function delta_l_i */
/* delta_l(i+1/2,j) */
/*************************************************************/


float delta_l_i(r,theta,i,j)
float r[][J_MAX], theta[];
int i,j;

{
        float x1,x2,x3,x4,x5,x6,x7;
        int j1,j2;
        perror(i,j);
        j1 = (j % JJ)+1;
        if ( j == 1 ) { j2 = JJ; }
        else { j2 = j - 1; }

        x2 = upperr(r,i,j)-lowerr(r,i,j);
        x3 = x2 * x2;
        x7 = subt(theta,j1,j2);
        x4 = x7*x7;
/*x4 = (float)fabs(theta[j1]-theta[j2])*(float)fabs(theta[j1]-theta[j2]); */
        x5 = x4 * upperr(r,i,j) * lowerr(r,i,j) / 4.0 ;
        x6 = x3 + x5;
        x1 = (float)sqrt((double)x6);          /* do cast */
        return(x1);

} /* delta_l_i */
/*************************************************************/
/* function delta_l_j */
/* delta_l(i,j+1/2) */
/*************************************************************/
```

```
Apr  1 13:14 1987  delta.c Page 3


float delta_l_j(r,theta,i,j)
float r[][J_MAX],theta[];
int i,j;

{
        float x1,x2,x3,x4,x5,x7;
        int j1,j2;
        perror(i,j);
        j1 = (j % JJ)+1;
        if ( j == 1 ) { j2 = JJ; }
        else { j2 = j - 1; }

        x2 = (r[i][j1]-r[i][j]) * (r[i][j1]-r[i][j]);
/* x3 = (float)fabs(theta[j1]-theta[j]) * (float)fabs(theta[j1]-theta[j]); */
        x7 = subt(theta,j1,j);
        x3 = x7 * x7;
        x4 = x3 * r[i][j] * r[i][j1];
        x5 = x2 + x4;
        x1 = (float)sqrt((double)x5);          /* do cast */
        return(x1);

} /* delta_l_j */
/*************************************************************/
/* this is lower r(i,j) function */
/* r(i+1/2,j-1/2) */
/*************************************************************/


float lowerr(r,i,j)
float r[][J_MAX];
int i,j;

{
        float x1;
        int j1,j2;
        perror(i,j);
        j1 = (j % JJ)+1;
        if ( j == 1 ) { j2 = JJ; }
        else { j2 = j - 1; }

        x1 = (r[i][j2]+r[i][j]+r[i+1][j2]+r[i+1][j])/4.0;
        return(x1);

} /* lowerr */
/*************************************************************/
/* this is upper r(i,j) function */
/* r(i+1/2,j+1/2) */
/*************************************************************/


float upperr(r,i,j)
float r[][J_MAX];
int i,j;

{
        int j1,j2;
```

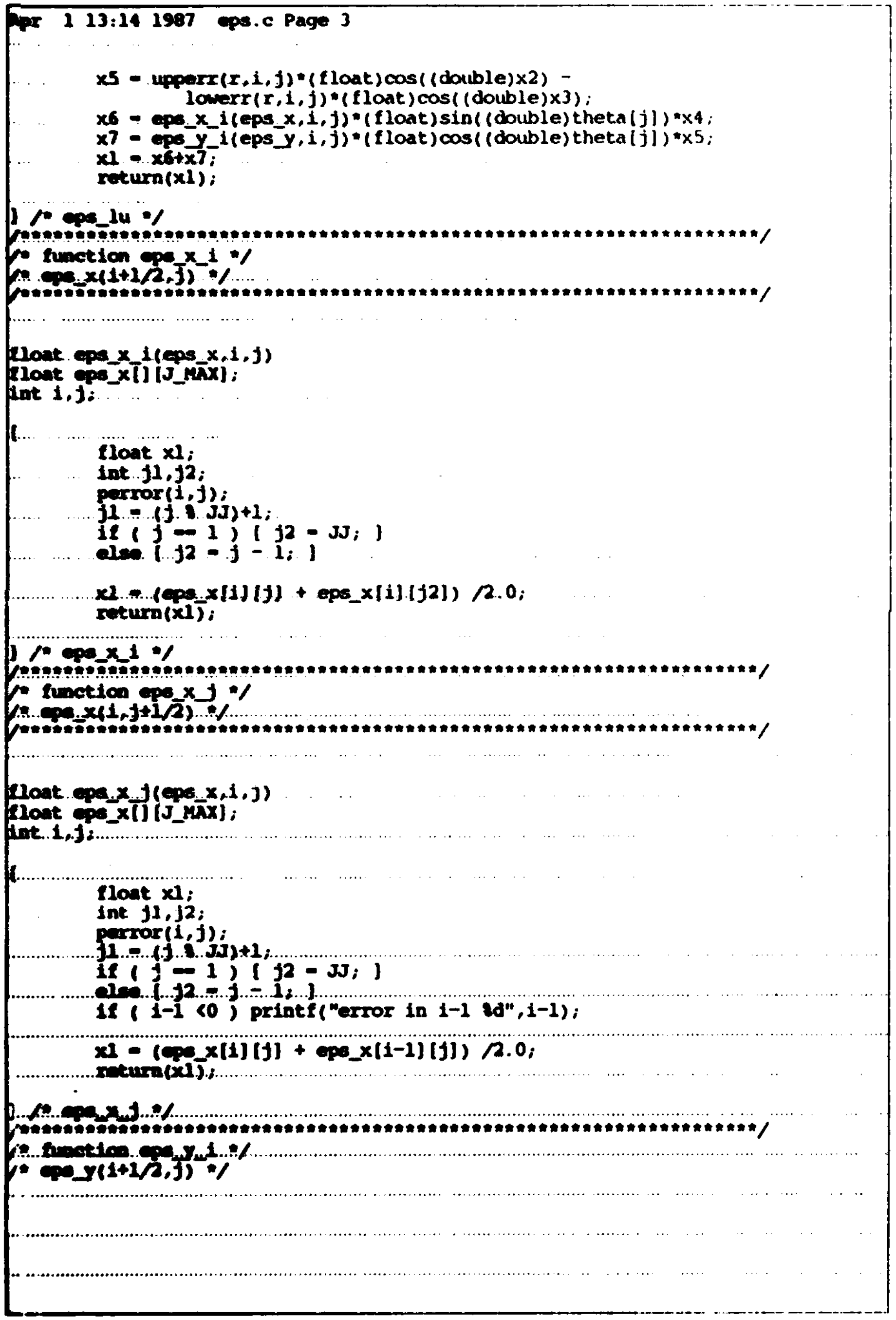
```
Apr  1 13:14 1987  delta.c Page 4


        perror(i,j);
        j1 = (j % JJ)+1;
        if ( j == 1 ) { j2 = JJ; }
        else { j2 = j - 1; }

        return((r[i][j]+r[i][j1]+r[i+1][j]+r[i+1][j1])/4.0);

} /* upperr */
/********************************************************************

this function test if error occurs in subscripts
if i or j are negative value , then print error and exit(1)
********************************************************************/

perror(i,j)
int i,j;

{
        if ( i <= -1 || j <= -1 ) {
                printf("subscript error occurred ,%d%\t%d%\n",i,j);
                exit(1);
        }
} /* perror */

/****************************************************************
*                       subt                                    *
* parameter :   theta, jmax ,jmin                               *
* function  :   add 2PI return if jmax < jmin                   *
****************************************************************/
float subt(theta,jmax,jmin)

float theta[];
int jmax,jmin;          /* upper and lower subscript */

{
        float x1;
        x1 = theta[jmax]-theta[jmin];
        if ( jmax < jmin )
                x1 = x1 + 2*M_PI;
        if ( x1 <0 )
        printf("ERROR IN THETA SUB  jmax = %d , jmin = %d %\n",jmax,jmin);
        return(x1);
} /* subt */
/****************************************************************
*                       addt                                    *
* parameter :   theta, jmax ,jmin                               *
* function  :   add 2PI return if jmax < jmin                   *
****************************************************************/
float addt(theta,jmax,jmin)

float theta[];
int jmax,jmin;          /* upper and lower subscript */

{
        float x1;
        x1 = theta[jmax] + theta[jmin];
```

```
Apr  1 13:14 1987  delta.c Page 5


        if ( jmax < jmin )
                x1 = x1 + 2*M_PI;
        if ( x1 <0 )
        printf("ERROR IN THETA SUB  jmax = %d , jmin = %d %\n",jmax,jmin);
        return(x1);
} /* addt */
```

```
Apr  1 13:14 1987  eps.c Page 1


/****************************************************************/
/* function eps_alpha */
/* eps_alpha(i,j+1/2) */
/****************************************************************/
#include "util.h"

float eps_alpha(r,theta,eps_x,eps_y,i,j)
float r[][J_MAX] , eps_x[][J_MAX] , eps_y[][J_MAX] , theta[];
int i,j;

{
        float x1,x2,x3,x4,x5;
        int j1,j2;
        perror(i,j);
        j1 = (j % JJ)+1;
        if ( j == 1 ) { j2 = JJ; }
        else { j2 = j - 1; }

/*      x2 = (float)fabs(theta[j1]+theta[j])/2; */
        x2 = addt(theta,j1,j) / 2.0;
        x3 = (float)sin((double)x2) * (float)sin((double)x2);
        x4 = (float)cos((double)x2) * (float)cos((double)x2);
        x1 = eps_x_j(eps_x,i,j) * x3 + eps_y_j(eps_y,i,j) * x4;
        return(x1);

} /* eps_alpha */
/****************************************************************/
/* function eps_beta */
/* eps_beta(i,j+1/2) */
/****************************************************************/

float eps_beta(r,theta,eps_x,eps_y,i,j)
float r[][J_MAX] , eps_x[][J_MAX] , eps_y[][J_MAX] , theta[];
int i,j;

{
        float x1,x2,x3,x4,x5,x6,x7,x8;
        int j1,j2;
        perror(i,j);
        j1 = (j % JJ)+1;
        if ( j == 1 ) { j2 = JJ; }
        else { j2 = j - 1; }
        x2 = addt(theta,j1,j)/2.0;
/*      x2 = (float)fabs(theta[j1]+theta[j])/2.0; */
        x3 = (float)sin((double)x2);
        x4 = (float)cos((double)x2);
        x5 = r[i][j1]*(float)sin((double)theta[j1]) -
            r[i][j]*(float)sin((double)theta[j]);
        x6 = r[i][j1]*(float)cos((double)theta[j1]) -
            r[i][j]*(float)cos((double)theta[j]);
        x7 = eps_x_j(eps_x,i,j);
        x8 = eps_y_j(eps_y,i,j);
        x1 = x7*x5*x3 + x8*x6*x4;
        return(x1);

} /* eps_beta */
```

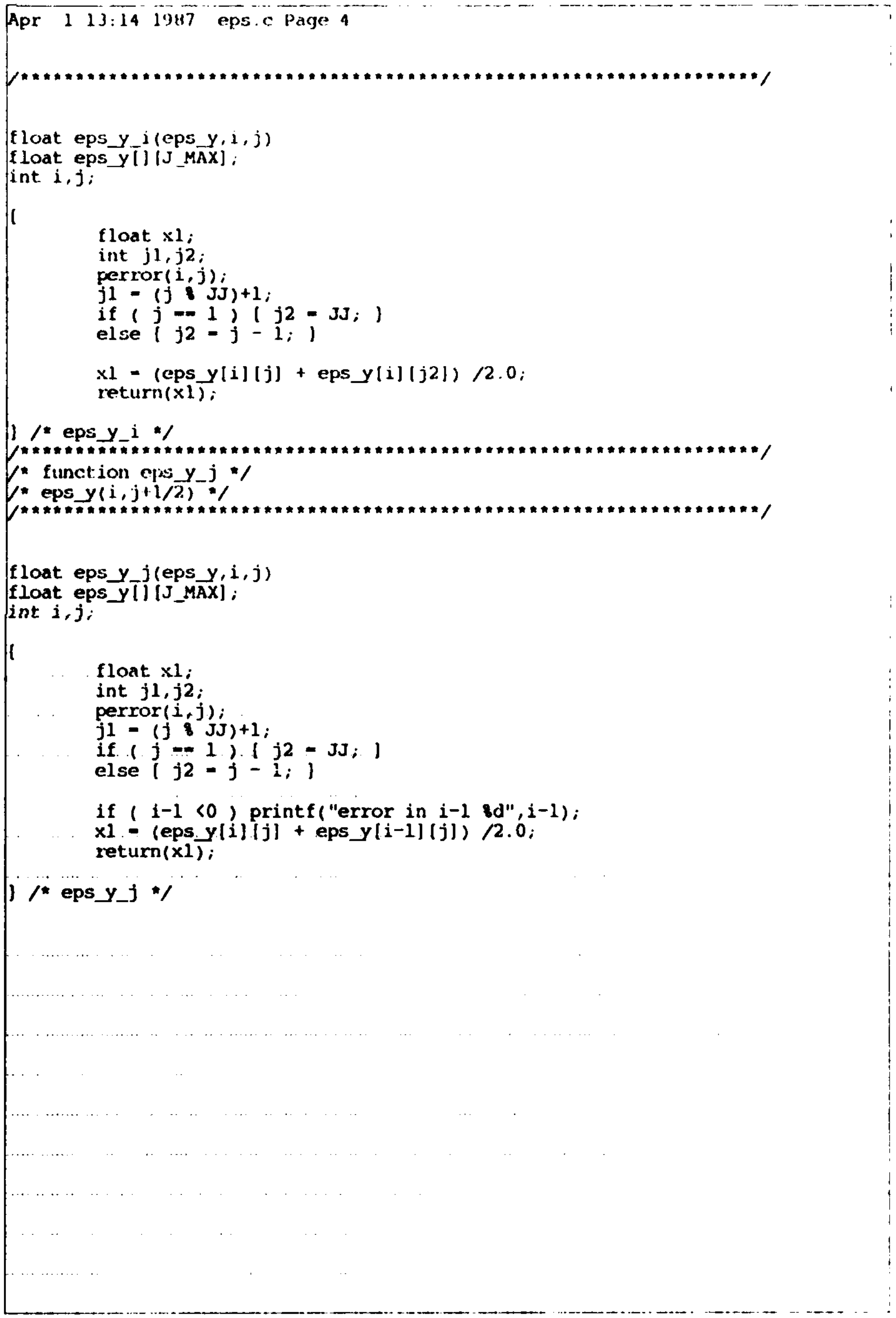
-29-

```
Apr  1 13:14 1987  eps.c Page 2


/****************************************************************/
/* function eps_lamda */
/* eps_lamda(i+1/2,j) */
/****************************************************************/


float eps_lamda(r,theta,eps_x,eps_y,i,j)
float r[][J_MAX] , eps_x[][J_MAX] , eps_y[][J_MAX] , theta[];
int i,j;

{
        float x1,x2,x3,x4,x5,x6,x7,x8,x9,x10,x11,x12,x13;
        int j1,j2;
        perror(i,j);
        j1 = (j % JJ)+1;
        if ( j == 1 ) { j2 = JJ; }
        else { j2 = j - 1; }

        x2 = addt(theta,j1,j) / 2.0;
        x3 = addt(theta,j,j2) / 2.0;
/*      x2 = (theta[j1]+theta[j]) / 2.0; */
/*      x3 = (theta[j]+theta[j2]) / 2.0; */
        x4 = (float)sin((double)x2);
        x5 = (float)cos((double)x2);
        x6 = (float)sin((double)x3);
        x7 = (float)cos((double)x3);
        x8 = upperr(r,i,j)*x4 - lowerr(r,i,j)*x6;
        x9 = upperr(r,i,j)*x5 - lowerr(r,i,j)*x7;
        x1 = eps_x_i(eps_x,i,j)*x8*x8 + eps_y_i(eps_y,i,j)*x9*x9;
        return(x1);

} /* eps_lamda */
/****************************************************************/
/* function eps_lu */
/* eps_lu(i+1/2,j) */
/****************************************************************/


float eps_lu(r,theta,eps_x,eps_y,i,j)
float r[][J_MAX] , eps_x[][J_MAX] , eps_y[][J_MAX] , theta[];
int i,j;

{
        float x1,x2,x3,x4,x5,x6,x7;
        int j1,j2;
        perror(i,j);
        j1 = (j % JJ)+1;
        if ( j == 1 ) { j2 = JJ; }
        else { j2 = j - 1; }

        x2 = addt(theta,j1,j) / 2.0;
        x3 = addt(theta,j,j2) / 2.0;
/*      x2 = (theta[j1]+theta[j])/2.0; */
/*      x3 = (theta[j]+theta[j2])/2.0; */
        x4 = upperr(r,i,j)*(float)sin((double)x2) -
                lowerr(r,i,j)*(float)sin((double)x3);
```



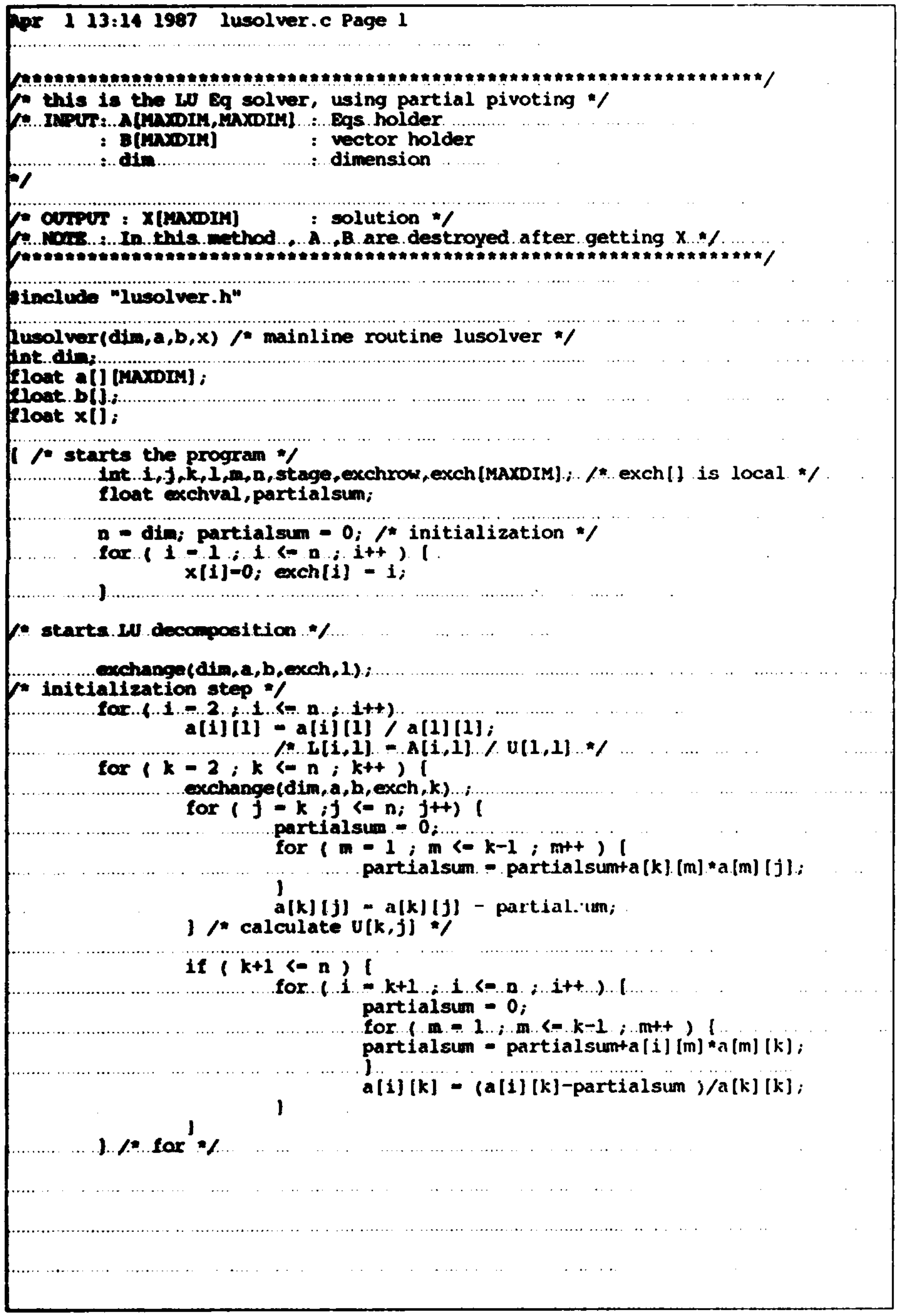
Apr 1 13:14 1987 eps.c Page 3

```
        x5 = upperr(r,i,j)*(float)cos((double)x2) -
             lowerr(r,i,j)*(float)cos((double)x3);
        x6 = eps_x_i(eps_x,i,j)*(float)sin((double)theta[j])*x4;
        x7 = eps_y_i(eps_y,i,j)*(float)cos((double)theta[j])*x5;
        x1 = x6+x7;
        return(x1);

} /* eps_lu */
/*****************************************************************/
/* function eps_x_i */
/* eps_x(i+1/2,j) */
/*****************************************************************/


float eps_x_i(eps_x,i,j)
float eps_x[][J_MAX];
int i,j;

{
        float x1;
        int j1,j2;
        perror(i,j);
        j1 = (j % JJ)+1;
        if ( j == 1 ) { j2 = JJ; }
        else { j2 = j - 1; }

        x1 = (eps_x[i][j] + eps_x[i][j2]) /2.0;
        return(x1);

} /* eps_x_i */
/*****************************************************************/
/* function eps_x_j */
/* eps_x(i,j+1/2) */
/*****************************************************************/


float eps_x_j(eps_x,i,j)
float eps_x[][J_MAX];
int i,j;

{
        float x1;
        int j1,j2;
        perror(i,j);
        j1 = (j % JJ)+1;
        if ( j == 1 ) { j2 = JJ; }
        else { j2 = j - 1; }
        if ( i-1 <0 ) printf("error in i-1 %d",i-1);

        x1 = (eps_x[i][j] + eps_x[i-1][j]) /2.0;
        return(x1);

} /* eps_x_j */
/*****************************************************************/
/* function eps_y_i */
/* eps_y(i+1/2,j) */
```

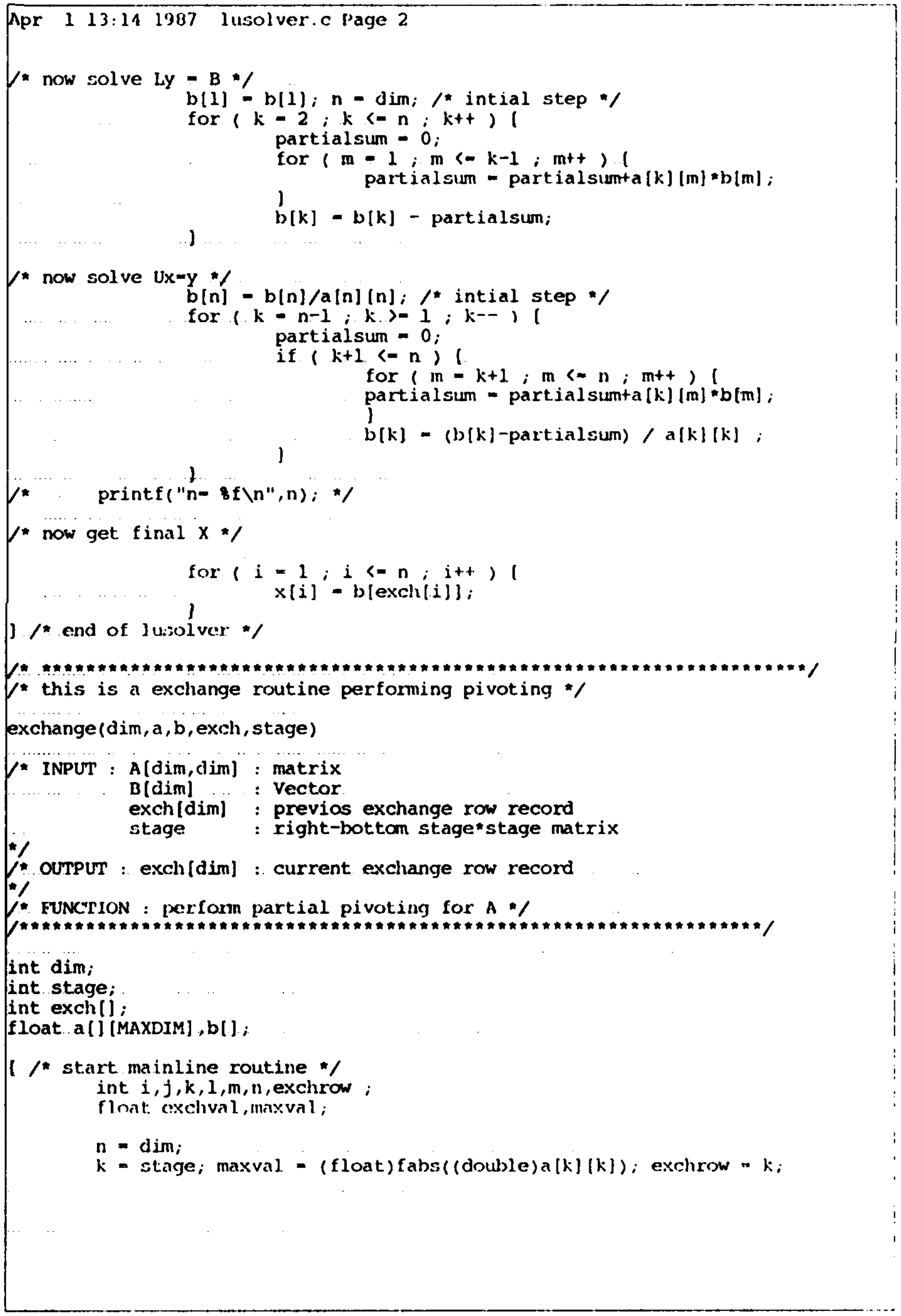


Apr 1 13:14 1987 eps.c Page 4

```
/*****************************************************************/


float eps_y_i(eps_y,i,j)
float eps_y[][J_MAX];
int i,j;

{
        float x1;
        int j1,j2;
        perror(i,j);
        j1 = (j % JJ)+1;
        if ( j == 1 ) { j2 = JJ; }
        else { j2 = j - 1; }

        x1 = (eps_y[i][j] + eps_y[i][j2]) /2.0;
        return(x1);

} /* eps_y_i */
/*****************************************************************/
/* function eps_y_j */
/* eps_y(i,j+1/2) */
/*****************************************************************/


float eps_y_j(eps_y,i,j)
float eps_y[][J_MAX];
int i,j;

{
        float x1;
        int j1,j2;
        perror(i,j);
        j1 = (j % JJ)+1;
        if ( j == 1 ) { j2 = JJ; }
        else { j2 = j - 1; }

        if ( i-1 <0 ) printf("error in i-1 %d",i-1);
        x1 = (eps_y[i][j] + eps_y[i-1][j]) /2.0;
        return(x1);

} /* eps_y_j */
```



```
Apr  1 13:14 1987  lusolver.c Page 1

/******************************************************************/
/* this is the LU Eq solver, using partial pivoting */
/* INPUT: A[MAXDIM,MAXDIM] : Eqs holder
        : B[MAXDIM]        : vector holder
        : dim              : dimension
*/

/* OUTPUT : X[MAXDIM]      : solution */
/* NOTE : In this method , A ,B are destroyed after getting X */
/******************************************************************/

#include "lusolver.h"

lusolver(dim,a,b,x) /* mainline routine lusolver */
int dim;
float a[][MAXDIM];
float b[];
float x[];

{ /* starts the program */
        int i,j,k,l,m,n,stage,exchrow,exch[MAXDIM]; /* exch[] is local */
        float exchval,partialsum;

        n = dim; partialsum = 0; /* initialization */
        for ( i = 1 ; i <= n ; i++ ) {
                x[i]=0; exch[i] = i;
        }

/* starts LU decomposition */

        exchange(dim,a,b,exch,1);
/* initialization step */
        for ( i = 2 ; i <= n ; i++)
                a[i][1] = a[i][1] / a[1][1];
                        /* L[i,1] = A[i,1] / U[1,1] */
        for ( k = 2 ; k <= n ; k++ ) {
                exchange(dim,a,b,exch,k) ;
                for ( j = k ;j <= n; j++) {
                        partialsum = 0;
                        for ( m = 1 ; m <= k-1 ; m++ ) {
                                partialsum = partialsum+a[k][m]*a[m][j];
                        }
                        a[k][j] = a[k][j] - partial.um;
                } /* calculate U[k,j] */

                if ( k+1 <= n ) {
                        for ( i = k+1 ; i <= n ; i++ ) {
                                partialsum = 0;
                                for ( m = 1 ; m <= k-1 ; m++ ) {
                                partialsum = partialsum+a[i][m]*a[m][k];
                                }
                                a[i][k] = (a[i][k]-partialsum )/a[k][k];
                        }
                }
        } /* for */
```



```
Apr  1 13:14 1987  lusolver.c Page 2

/* now solve Ly = B */
                b[1] = b[1]; n = dim; /* intial step */
                for ( k = 2 ; k <= n ; k++ ) {
                        partialsum = 0;
                        for ( m = 1 ; m <= k-1 ; m++ ) {
                                partialsum = partialsum+a[k][m]*b[m];
                        }
                        b[k] = b[k] - partialsum;
                }

/* now solve Ux=y */
                b[n] = b[n]/a[n][n]; /* intial step */
                for ( k = n-1 ; k >= 1 ; k-- ) {
                        partialsum = 0;
                        if ( k+1 <= n ) {
                                for ( m = k+1 ; m <= n ; m++ ) {
                                partialsum = partialsum+a[k][m]*b[m];
                                }
                                b[k] = (b[k]-partialsum) / a[k][k] ;
                        }
                }
/*      printf("n= %f\n",n); */

/* now get final X */

                for ( i = 1 ; i <= n ; i++ ) {
                        x[i] = b[exch[i]];
                }
} /* end of lusolver */

/* ****************************************************************/
/* this is a exchange routine performing pivoting */

exchange(dim,a,b,exch,stage)

/* INPUT : A[dim,dim] : matrix
           B[dim]     : Vector
           exch[dim]  : previos exchange row record
           stage      : right-bottom stage*stage matrix
*/
/* OUTPUT : exch[dim] : current exchange row record
*/
/* FUNCTION : perform partial pivoting for A */
/*****************************************************************/

int dim;
int stage;
int exch[];
float a[][MAXDIM],b[];

{ /* start mainline routine */
        int i,j,k,l,m,n,exchrow ;
        float exchval,maxval;

        n = dim;
        k = stage; maxval = (float)fabs((double)a[k][k]); exchrow = k;
```

Apr 1 13:14 1987 lusolver.c Page 3

```
/* initialzation step */
        while ( ++k <= dim ) {
                if ( (float)fabs((double)a[k][stage]) > maxval ) {
                        exchrow = k;
                        maxval = (float)fabs((double)a[k][stage]);
                }
        }

        if ( exchrow != stage ) { /* exchange occurs */
                m = exch[stage];
                exch[stage] = exch[exchrow];
                exch[exchrow] = m;
/* record exchanges */
                for ( j = 1 ; j <= n ; j++ ) {
                        exchval = a[exchrow][j];
                        a[exchrow][j] = a[stage][j];
                        a[stage][j] = exchval;
                } /* exchange row in A */

                exchval = b[exchrow];
                b[exchrow] = b[stage];
                b[stage] = exchval;
        } /* perfrom exchange */
} /* end of exchange */

/*****************************************************************/
/* TEST MATRIX AND VECTOR */
/*****************************************************************/

test(a,b,diml)
int diml;
float a[][MAXDIM];
float b[];
{       int i,j,n;

        n = diml;
        printf("n=,%f\n",n);
        for ( i = 1 ; i <= n ; i++ ) {
                for ( j = 1 ; j <= n ; j++ ) {
                        printf("%f\n",a[i][j]);
                }
        }
        for ( i = 1 ; i <= n ; i++ ){
                printf("%f\n",b[i]);
        }
}
```

Apr 1 13:14 1987 mainline.c Page 1

```
/*******************************************************************
This is main program , which first defines all global variables,
arrays, and then reads all user defined material properties, grid
points and time step and step size.
User have to provide the program Jx and Jy functions in R3

Algorithm :
1) read all grid points, fill out arrays r[][J MAX],theta[J MAX] with
   regular grid points in R2 & R3.
2) read all material properties ,eps_x[][J_MAX],eps_y[][J_MAX];
3) read all functions Jx , Jy in R3.
4) initialize all e-field and h-field to 0.
5) loop
        first calculate all h-field through R1 --- R3;
        dump out h-old to disk;
        copy h-new to h-old to save data;
        then calculate all e-field through R1 --- R3;
        dump out e-old to disk;
        copy e-new to e-old to save data for next iteration;
   until ( step => TIME_STEP )
6) check errors in any

*******************************************************************/
#include "main.h"
/*******************************************************************/
/* define all extern data in "*.h" files */

/* I also need serveral subroutine which I include their names here,
        NOTE that I should define their name in "main.h"
*/
/*******************************************************************/
main(argc,argv)
/* begin main program */
int argc;
char *argv[];

{
        int total_level;        /* to be used instead of TIME_STEP */
        int level[TIME_STEP+10];        /* THE ARRAY HOLDS OUTPUT LEVEL */
        int irp,ithetap;                /* hold grid value of rp,thetap */

        Init_file(argc,argv);   /* open all files for input/output */
        printf("enter the source point , rp = ? thetap = ? %\n");
        scanf("%d%d", &irp, &ithetap);
        level_out(level);       /* read output level array */
/*      grid(r,theta); */
        prop(eps_x,eps_y,mu_z,sigma,&eps_0,&mu_0,&sigma_0);
        readgrid(r,theta);
        printf("read how many replications are needed , total_level %\n");
        scanf("%d", &total_level);
        init_energy_level();            /* read energy level to print */

        /* get real source address in theta and r */
        rp = irp;
        thetap = theta[ithetap]+(theta[ithetap+1]-theta[ithetap])/2.0;
        headpr(irp,ithetap);
```

```
Apr  1 13:14 1987  mainline.c Page 2


        /* critical point in rp = 16 , thetap = half_theta */
        /* read all material properties in R1, R2, R3 */

        n = 0;
        while ( n <= total_level ) {
/*      printf(" level = %d%\n ",n); */ /* interactive print level */
                h_field();                              /* get H - field */
/*              if ( in_level(n,level) ) {
                        dump_h();
                }
*/
                e_field();                              /* get E - field */
/*              if ( in_level(n,level) ) {
                        dump_e();
                }
*/
                copy_h();               /* copy h-new to h-old */
                copy_e();               /* copy e-new to e-old */
/* ---------------- statistical procedure ----------------------*/
                e_field_stat();
                if ( in_level(n,level) )
                        e_field_stat_pr();
                n++;            /* 1/2 vs. 1 in time step */

        } /* while */

} /* main */
/*****************************************************************/
/* this routine only reads material properties */
/* INPUT : eps_x[EPS][J_MAX] , eps_y[EPS][J_MAX],mu_z[EPS][J_MAX],
        sigma[EPS][J_MAX]

*/
/*****************************************************************/

readprop(eps_x,eps_y,mu_z,sigma)
float eps_x[][J_MAX] , eps_y[][J_MAX] , mu_z[][J_MAX] , sigma[][J_MAX];

{
        int i1, j1 ;
        /* in R1 , read these matrix */
        readmatrix(I_START,MM,JJ,eps_x);
        readmatrix(I_START,MM,JJ,eps_y);
        readmatrix(I_START,MM,JJ,mu_z );
        readmatrix(I_START,MM,JJ,sigma);

        /* in R2 and R3 , read following and assign them to arrays */
        fscanf(fin,"%e%e%e",eps_0,mu_0,sigma_0);

} /* readprop */
/**************************************************************
 this is a matrix reading subroutine
**************************************************************/
readmatrix(start,end,jend,name)
```

```
Apr  1 13:14 1987  mainline.c Page 3

int start,end;
float name[][J_MAX];

{
        int i1,j1;
        for ( i1 = start ; i1 <= end ; i1++ ) {
                for ( j1 = 1 ; j1 <= jend ; j1++ ) {
                        fscanf(fin,"%e",&name[i1][j1]);
                }
        }
        return;
} /* readmatrix */
/*******************************************************************
 This Function reads grids r[i][j], theta[j]
 INPUT : r[I_MAX][J_MAX]
         : theta[J_MAX]
*******************************************************************/

readgrid(r,theta)
float r[][J_MAX];
float theta[];

{ /* start routine */
        int i1, j1;
        float deg;

        /* AT circle origin , we define r[i][j] to be 0 */
        for ( j1 = 1 ; j1 <= JJ ; j1++ ) {
                r[0][j1] = 0;
        }
        for ( i1 = 1 ; i1 <= II ; i1++ ) {
                for ( j1 = 1 ; j1 <= JJ ; j1++ ) {
                        fscanf(fin,"%e",&r[i1][j1]);
/*                      fprintf(fout,"i= %d j= %d r= %e\n", i1,j1,r[i1][j1]);*/
                }
        }

        for ( j1 = 1 ; j1 <= JJ ; j1++ ) {
                fscanf(fin,"%e",&deg);
                theta[j1] = deg * M_PI / 180.0;
/*              fprintf(fout,"j= %d theta = %e\n",j1,theta[j1]); */
        }
        for ( i1 = II+1 ; i1 < I_MAX ; i1++ ) {
                for ( j1 = 1 ; j1 <= JJ ; j1 ++ ) {
                        r[i1][j1] = r[II][j1];
                }
        }

} /* readgrid */
/*******************************************************************/
/* this function dump all e_field or H_field to disk */
/*******************************************************************/

dump_h()
{
        int i,j,k,l;
        int k1,k2,k3,k4,k5;
        int ksub[10];

        fprintf(fout, "************************************ level n = %d\n",
                n);

        for ( i = 15 ; i <= II ; i++ ) {
                fprintf(fout,"h field ,i= %d%\n",i);
                for ( k1 = 1 ; k1 <= 8 ; k1++ ) {
                        for ( k2 = 1;k2 <= 4 ; k2++ ) {
                          ksub[k2] = (k1-1)*4 + k2;
                        }
                        fprintf(fout,"%e  %e  %e  %e%\n",
        h_new[i][ksub[1]],h_new[i][ksub[2]],h_new[i][ksub[3]],
        h_new[i][ksub[4]]);
                }
        }

} /* dump_h */

/****************************************************************/
/*      this function dump all e_field to disk for reading       */
/****************************************************************/
dump_e()
{
        int i,j,k,l;
        int k1,k2,k3,k4,k5;
        int ksub[10];

        fprintf(fout, "************************************ level n = %d\n",
                n);
        for ( i = 15 ; i <= II ; i++ ) {
                fprintf(fout,"integer e_field ,i= %d---------------%\n",i);
                for ( k1 = 1 ; k1 <= 8 ; k1++ ) {
                        for ( k2 = 1;k2 <= 4 ; k2++ ) {
                          ksub[k2] = (k1-1)*4 + k2;
                        }
                        fprintf(fout,"%e  %e  %e  %e%\n",
        ej_new[i][ksub[1]],ej_new[i][ksub[2]],ej_new[i][ksub[3]],
        ej_new[i][ksub[4]]);
                }
                fprintf(fout,"half e_field ,i= %d $$$$$$$$$$$$$$$$$$$$%\n",i);
                for ( k1 = 1 ; k1 <= 8 ; k1++ ) {
                        for ( k2 = 1;k2 <= 4 ; k2++ ) {
                          ksub[k2] = (k1-1)*4 + k2;
                        }
                        fprintf(fout,"%e  %e  %e  %e%\n",
        ei_new[i][ksub[1]],ei_new[i][ksub[2]],ei_new[i][ksub[3]],
        ei_new[i][ksub[4]]);
                }
        }

} /* dump_e */

/****************************************************************/
```

```
/* these functions copy e-field and h-field */
/****************************************************************/
copy_e()
{
        int i,j,k,l;

        for ( i = 0 ; i <= II ; i++ ) {
                for ( j = 1 ; j <= JJ ; j++ ) {
                        ei_old[i][j] = ei_new[i][j];
                        ej_old[i][j] = ej_new[i][j];
                }
        }

} /* copy_e */

/****************************************************************/
/*      This function : copy h_field to generate new step computation */
/****************************************************************/
copy_h()
{
        int i,j,k,l;

        for ( i = 0 ; i <= II ; i++ ) {
                for ( j = 1 ; j <= JJ ; j++ ) {
                        h_old[i][j] = h_new[i][j];
                }
        }

} /* copy_h */
/****************************************************************/
/* these functions actually calculates the e-field , h-field in ONE level */
/****************************************************************/

e_field()

{
        int i,j,k,l,m;
        float x1,x2,x3,x4,x5,x6,x7x8;

        /* i = 0 , internal boundary case */
        i = 0;
        for ( j = 1 ; j <= JJ ; j++ ) {
get_ei_0(h_new,h_old,ei_old, ej_old,ei_new,ej_new,r,mu_z, theta ,i,j,n,t);
        }
        /* i = 1 .. MM , i.e, in R1  */
        for ( i = 1 ; i <= MM ; i++ ) {
                for ( j = 1 ; j <= JJ ; j++ ) {
                        x1 = a1(r,theta,eps_x,eps_y,sigma,i,j,n,t);
                        x2 = a2(r,theta,eps_x,eps_y,sigma,i,j,n,t);
                        x3 = a3(r,theta,eps_x,eps_y,sigma,i,j,n,t);
                        x4 = a4(r,theta,eps_x,eps_y,sigma,i,j,n,t);
        x5 = b1(r,theta,eps_x,eps_y,sigma,ei_old,ej_old,h_new,h_old,i,j,n,t);
        x6 = b2(r,theta,eps_x,eps_y,sigma,ei_old,ej_old,h_new,h_old,i,j,n,t);
                        a[1][1] = x1; a[1][2] = x2;
                        a[2][1] = x3; a[2][2] = x4;
                        b[1] = x5;    b[2] = x6;
                        dim = 2;
                        lusolver(dim,a,b,x);    /* invoke solver to get x */
                        ej_new[i][j] = x[1];    /* get ej_new */
                        ei_new[i][j] = x[2];    /* get ei_new */
                }
        }

        /* i = MM+1 .. NN , i.e, in R2  */
        for ( i = MM+1 ; i <= NN ; i++ ) {
                for ( j = 1 ; j <= JJ ; j++ ) {
                        x1 = a5(r,theta,eps_0,mu_0,sigma_0,i,j,n,t);
                        x2 = a6(r,theta,eps_0,mu_0,sigma_0,i,j,n,t);
                        x3 = a7(r,theta,eps_0,mu_0,sigma_0,i,j,n,t);
                        x4 = a8(r,theta,eps_0,mu_0,sigma_0,i,j,n,t);
        x5 = b3(r,theta,eps_0,sigma_0,mu_0,ei_old,ej_old,h_new,h_old,i,j,n,t);
        x6 = b4(r,theta,eps_0,sigma_0,mu_0,ei_old,ej_old,h_new,h_old,i,j,n,t);
                        a[1][1] = x1; a[1][2] = x2;
                        a[2][1] = x3; a[2][2] = x4;
                        b[1] = x5;    b[2] = x6;
                        dim = 2;
                        lusolver(dim,a,b,x);    /* invoke solver to get x */
                        ej_new[i][j] = x[1];    /* get ej_new */
                        ei_new[i][j] = x[2];    /* get ei_new */
                }
        }
        /* i = NN+1 .. II-1 , i.e, in R3  */
        for ( i = NN+1 ; i <= II-1 ; i++ ) {
                for ( j = 1 ; j <= JJ ; j++ ) {
                        get_ej(r,theta,h_new,h_old,ej_new,ej_old,eps_0,
                        sigma_0,mu_0,i,j,n,t,rp,thetap);

                        get_ei(r,theta,h_new,h_old,ei_new,ei_old,eps_0,
                        sigma_0,mu_0,i,j,n,t,rp,thetap);
                }
        }

        /* external boundary comes */
        for ( j = 1 ; j <= JJ ; j++ ) {
                get_ej_bound(mu_0,eps_0,h_new,h_old,ej_new,ej_old,j);
        }

} /* e_field */

/*******************************************************************
   this function get h_field in R1, R2, R3 respectively
*******************************************************************/
h_field()

{
        int i,j,k,l;
        float x1,x2,x3,x4,x5,x6;

        /* i = 0 ; internal boundary conditions formula */
        for ( j = 1 ; j <= JJ ; j++ ) {
                get_h_0(j,n,t);
        }
```

```
        /* i = 1 .. MM , i.e. in R1 */
        for ( i = 1 ; i <= MM ; i++ ) {
                for ( j = 1 ; j <= JJ ; j++ ) {
                        h1(h_new,h_old,ei_old,ej_old,r,mu_z,theta,i,j,n,t);
                }
        }

        /* i = MM+1 .. NN,  i.e. in R2 */
        for ( i = MM+1 ; i<= NN ; i++ ) {
                for ( j = 1 ; j <= JJ ; j++ ) {
                        h2(h_new,h_old,ei_old, ej_old,r,mu_0, theta , i,j,n,t);
                }
        }

        /* i = NN+1 .. II-1 ,, i.e. in R3 */
        for ( i = NN+1 ; i <= II-1 ; i++ ) {
                for ( j = 1 ; j <= JJ ; j++ ) {
        h3(r,theta,eps_0,mu_0,sigma_0,h_new,h_old,ei_old,ej_old,i,j,n,t);
                }
        }

        /* exterior boundary comes */
        /* NO NEED TO CALCULATE */

} /* h_field */

Init_file(argc,argv)
/************************************************************************

                I N I T I A L I Z E _ F I L E S

This Procedure will Check if the Command String is valid, containing any
correctly specified options followed by the Input File name followed by an
Output File Name.  If no parameters are supplied, execution will cease
with exit(0).  If an invalid option sequence is supplied, execution will
stop with exit(1).  If no Input File is given or the specified file could
not be opened, an error message is displayed and execution stops with
exit(2).  If no Output file name is given "RDR.output" is assumed.  Note
that the default options are a black and white screen with display 1.  The
desired screen and display types are sent back in there respective
variables. In addition, if no errors were detected in the parameters, the
input file is opened for reading.

************************************************************************/
int argc;
char *argv[];

#define         BW              1    /* A Black and White Screen */
#define         COLOR           2    /* A Color Screen */
#define         CBW             3    /* Color Monitor on a B/W Screen */

{

 int i,c;
  if (argc==1)                  /* No Parameters */
```

Apr 1 13:14 1987 mainline.c Page 8

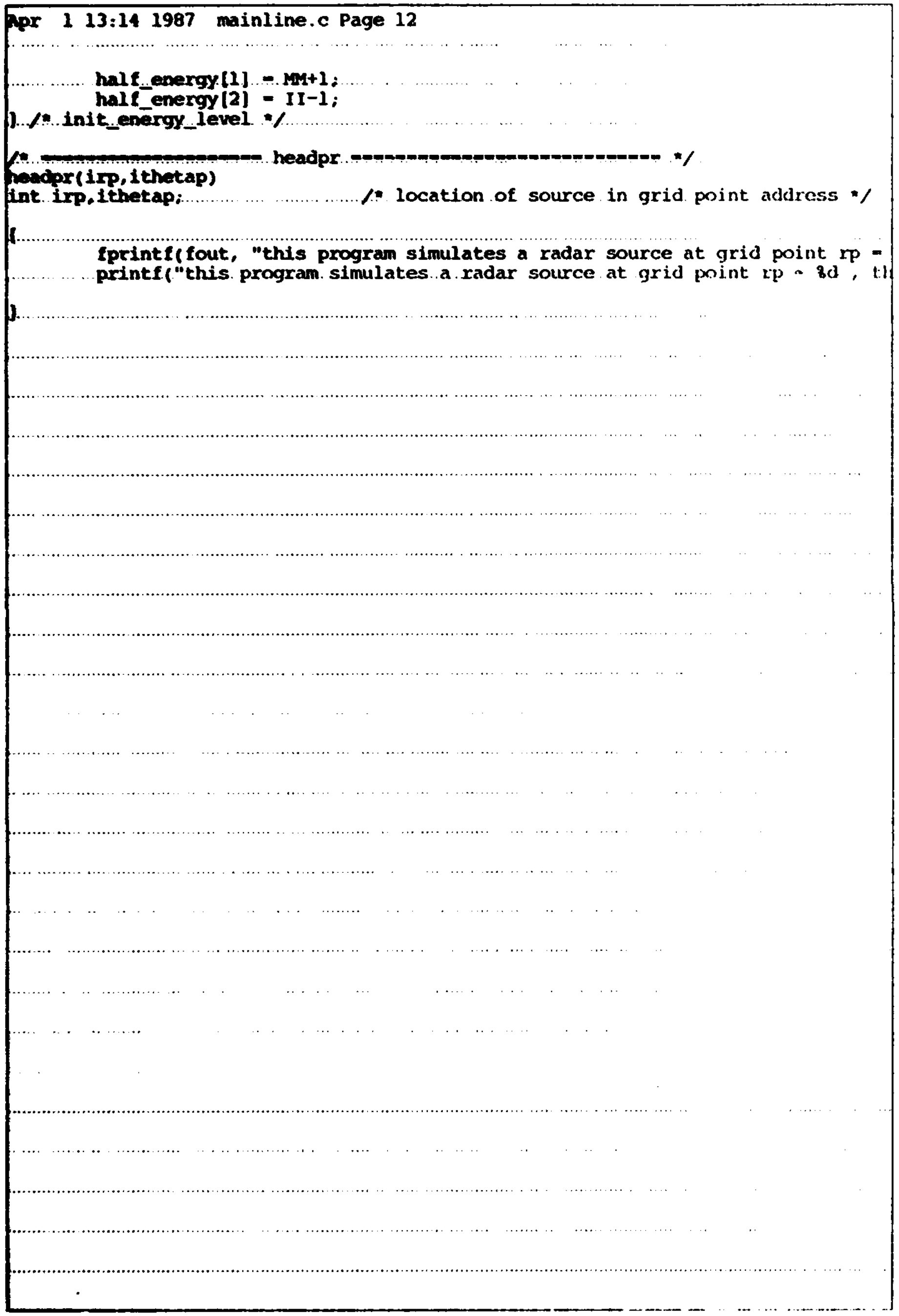
```
 {printf("Usage: rdr Input_File Output_File [Default RDR.output]\n");
    exit(0);
    }
  else
    { *++argv; c = 0;
    if (argc==1+c)                    /* No Input File Specified */
 {printf("Usage: rdr Input_File Output_File [Default RDR.output]\n");
      exit(2);
      }
    else if ((fin=fopen(*argv,"r"))==NULL)    /* Input File Not Accessable */
          { printf("rdr: could not open %s\n",*argv);
            exit(2);
          }

    else if (argc==2+c) strcpy(outfile,"RDR.output");         /* Default */
    else strcpy(outfile,*++argv);               /* Output File Specified */
         if((fout=fopen(outfile,"w"))==NULL)  /* Output File open Error */
           { printf("rdr: could not open %s\n",outfile);
           }
        }
}
/*******************************************************************
strcpy : copy string t to string s
*******************************************************************/
strcpy(s,t)                                /* copy t to s */
char *s,*t;
{
        while (*s++ = *t++ )
                ;
} /* strcpy */

/***************************************************************
This is chen's read function only for a circle.
the read is regular theta and regular radius.

***************************************************************/

grid(r,theta)
float r[][J_MAX], theta[];

{
        float x1,x2,x3;
        int i,j,k;

        for ( j = 1 ; j <= JJ ; j++ ) {
                theta[j] = 0.2618 * (j-1);
        }

        /* in R1 , get i = 1 .. MM */
        for ( i = 0 ; i <= MM ; i++ ) {
                for ( j = 1 ; j <= JJ ; j++ ) {
                        r[i][j] = i;
                }
        }

        /* in R2 , get i = MM+1 .. NN */
```

Apr 1 13:14 1987 mainline.c Page 9

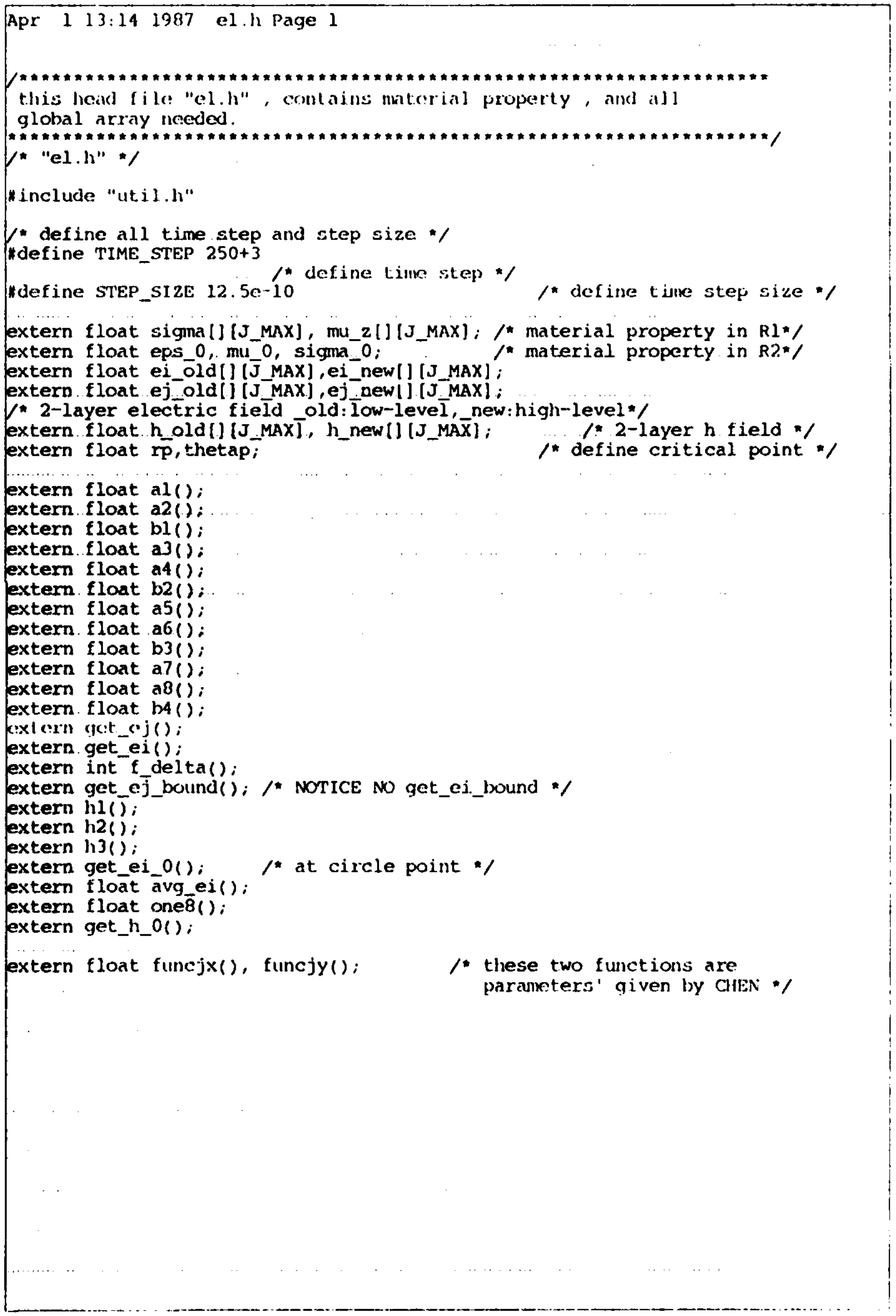
```
        for ( i = MM+1 ; i <= NN ; i++ ) {
                for ( j = 1 ; j <= JJ ; j++ ) {
                        r[i][j] = i;
                }
        }

        /* in R3 , get i = NN+1 .. II */
        for ( i = NN+1 ; i <= II ; i++ ) {
                for ( j = 1 ; j <= JJ ; j++ ) {
                        r[i][j] = i;
                }
        }
} /* grid */
/*******************************************************************
This routine reads all material points in R1 , R2 , R3
this is regular because of CHEN's circle.

*******************************************************************/

prop(eps_x,eps_y,mu_z,sigma,eps_0,mu_0,sigma_0)
float eps_x[][J_MAX] , eps_y[][J_MAX] , mu_z[][J_MAX] , sigma[][J_MAX];
float *eps_0, *mu_0, *sigma_0;

{
        int i,j;
        float x1,x2;

        /* in R1 , we need eps_x, eps_y, mu_z , sigma */
        for ( i = 0 ; i <= MM ; i++ ) {
                for ( j = 1 ; j <= JJ ; j++ ) {
                        eps_x[i][j] = 8.8419e-12;
                        eps_y[i][j] = 8.8419e-12;
                        mu_z[i][j] = 1.2566e-6;
                        sigma[i][j] = 4.0e7;
                }
        }

        /* in R2, and R3 we need eps_0, mu_0, sigma_0 */
/* for cross problem , I still need KNOW MM+1..EPS-1's points characters */
        for ( i = MM+1 ; i <= EPS-1 ; i++ ) {
                for ( j = 1 ; j <= JJ ; j++ ) {
                        eps_x[i][j] = 8.8419e-12;
                        eps_y[i][j] = 8.8419e-12;
                        mu_z[i][j] = 1.2566e-6;
                        sigma[i][j] = 1.0e-4;
                }
        }


        *eps_0 = 8.8419e-12;
        *mu_0  = 1.2566e-6;
        *sigma_0 = 1.0e-4;
} /* prop */
/*******************************************************************
this routine reads all levels on which CHEN wants to dump his E and
H fields
```

```
*****************************************************************/

level_out(level)
int level[];

{
        int il,jl,out_level,num_level;
        jl = 0;
        printf(" how many level do you want to print out ? %\n");
        scanf("%d",&num_level);
        for ( il = 0 ; il <= TIME_STEP+9 ; il++ ) {
                level[il] = 0;
        }
        for ( il = 1 ; il <= num_level ; il++ ) {
                printf("input the levels you want to output to disk %d%\n",il);
                scanf("%d",&level[jl++]);
        }
} /* level_out */

/****************************************************************
this routine ask if n is in the level[] array
*****************************************************************/

int in_level(n,level)
int n;
int level[];

{
        int i;

        for ( i = 0 ; i <= TIME_STEP ; i++ ) {
                if ( n == level[i] ) { return(TRUE);}
        }
        return(FALSE);
} /* in_level */

/* ================== statistical procedure =====================*/
e_field_stat()
{
        int i,j,k;
        float x1,x2;

        for ( j = 1 ; j <= JJ ; j++ ) {
                ej_stat[1][j] += ej_old[MM+1][j] * ej_old[MM+1][j];
        }
        for ( j = 1 ; j <= JJ ; j++ ) {
                ej_stat[2][j] += ej_old[II-1][j] * ej_old[II-1][j];
        }
        for ( j = 1 ; j <= JJ ; j++ ) {
                ej_stat[3][j] += ej_old[II][j] * ej_old[II][j];
        }
        for ( j = 1 ; j <= JJ ; j++ ) {
                ei_stat[1][j] += ei_old[MM+1][j] * ei_old[MM+1][j];
        }
        for ( j = 1 ; j <= JJ ; j++ ) {
                ei_stat[2][j] += ei_old[II-1][j] * ei_old[II-1][j];
```

```
            }
} /* e_field_stat */

/* ================= e_field_stat_pr ================= */
e_field_stat_pr()
{
        int i,j,k,l;
        int k1,k2,k3,k4,k5;
        int ksub[10];


        fprintf(fout, "************************************ level n = %d\n",
                n);
        fprintf(fout, " energy integer e_field : MM,II-1 II%\n",
                n);
        for ( i = 1 ; i <= 3 ; i++ ) {
           fprintf(fout,"energy integer ,i= %d----%\n",int_energy[i]);
                for ( k1 = 1 ; k1 <= 8 ; k1++ ) {
                        for ( k2 = 1;k2 <= 4 , k2++ ) {
                          ksub[k2] = (k1-1)*4 + k2;
                        }
                        fprintf(fout,"%e  %e  %e  %e%\n",
        ej_stat[i][ksub[1]],ej_stat[i][ksub[2]],ej_stat[i][ksub[3]],
        ej_stat[i][ksub[4]]);
                }
        }

        fprintf(fout, " energy half e_field : MM,II-1 II%\n",
                n);
        for ( i = 1 ; i <= 2 ; i++ ) {
          fprintf(fout,"energy half ,i= %d%\n",half_energy[i]);
                for ( k1 = 1 ; k1 <= 8 ; k1++ ) {
                        for ( k2 = 1;k2 <= 4 ; k2++ ) {
                          ksub[k2] = (k1-1)*4 + k2;
                        }
                        fprintf(fout,"%e  %e  %e  %e%\n",
        ei_stat[i][ksub[1]],ei_stat[i][ksub[2]],ei_stat[i][ksub[3]],
        ei_stat[i][ksub[4]]);
                }
        }

} /* e_field_stat_pr */

/* ==================== energy i , j ===================== */
init_energy_level()

{
        int i;

        for ( i = 0 ; i < 9 ; i++ ) {
                int_energy[i] = 0;
                half_energy[i] = 0;
        }
        int_energy[1] = MM+1;
        int_energy[2] = II-1;
        int_energy[3] = II;
```



```
Apr  1 13:14 1987  mainline.c Page 12


        half_energy[1] = MM+1;
        half_energy[2] = II-1;
} /* init_energy_level */

/* ================== headpr ========================== */
headpr(irp,ithetap)
int irp,ithetap;                    /* location of source in grid point address */

{
        fprintf(fout, "this program simulates a radar source at grid point rp =
        printf("this program simulates a radar source at grid point rp = %d , th

}
```



```
Apr  1 13:14 1987  el.h Page 1


/*****************************************************************
 this head file "el.h" , contains material property , and all
 global array needed.
******************************************************************/
/* "el.h" */

#include "util.h"

/* define all time step and step size */
#define TIME_STEP 250+3
                                        /* define time step */
#define STEP_SIZE 12.5e-10                                 /* define time step size */

extern float sigma[][J_MAX], mu_z[][J_MAX]; /* material property in R1*/
extern float eps_0, mu_0, sigma_0;          /* material property in R2*/
extern float ei_old[][J_MAX],ei_new[][J_MAX];
extern float ej_old[][J_MAX],ej_new[][J_MAX];
/* 2-layer electric field _old:low-level,_new:high-level*/
extern float h_old[][J_MAX], h_new[][J_MAX];          /* 2-layer h field */
extern float rp,thetap;                                /* define critical point */

extern float a1();
extern float a2();
extern float b1();
extern float a3();
extern float a4();
extern float b2();
extern float a5();
extern float a6();
extern float b3();
extern float a7();
extern float a8();
extern float b4();
extern get_ej();
extern get_ei();
extern int f_delta();
extern get_ej_bound(); /* NOTICE NO get_ei_bound */
extern h1();
extern h2();
extern h3();
extern get_ei_0();     /* at circle point */
extern float avg_ei();
extern float one8();
extern get_h_0();

extern float funcjx(), funcjy();        /* these two functions are
                                           parameters' given by CHEN */
```

```
Apr  1 13:14 1987  lusolver.h Page 1

/***************************************************************
 this head is to define some const and matrix of lusolver.c

***************************************************************/
/* "lusolver.h" */

#include <stdio.h>
#include <math.h>

#define MAXDIM 8                /* maximum dimension */
#define CONTROL 1e-8            /* error control parameter */

extern int  dim;                /* declaration of A's dimension */
extern float a[][MAXDIM];       /* matrix A holding Eqs */
extern float b[];               /* vector B holding right vector */
extern float x[];               /* vector X holding unknowns */
/* function declarations */
extern  lusolver();
extern  exchange();
extern  test();
```

```
Apr  1 13:14 1987  main.h Page 1

 /***************************************************************
  This head file contains all extern array defined in "util.h"
  , "el.h". there are some new defined arrays, include "lusolver.h"

***************************************************************/
/* "main.h" */

#include "el.h"                 /* which in turn including "util.h" */
#include "lusolver.h"

/* this time I do all initialization in compile time */

/* external definitions in "el.h" */

float sigma[EPS][J_MAX] = 0 , mu_z[EPS][J_MAX] = 0 ;
/* material property in R1*/
float eps_0 = 0, mu_0 = 0 , sigma_0 = 0;    /* material property in R2*/
float ei_old[I_MAX][J_MAX] = 0 ,ei_new[I_MAX][J_MAX] = 0;
float ej_old[I_MAX][J_MAX] = 0 ,ej_new[I_MAX][J_MAX] = 0;
/* 2-layer electric field _old:low-level,_new:high-level*/
float h_old[I_MAX][J_MAX] = 0 , h_new[I_MAX][J_MAX] = 0; /* 2-layer h field */
float rp = 0 ,thetap = 0;                        /* define critical point */
float a[MAXDIM][MAXDIM] = 0, b[MAXDIM] =0 ,x[MAXDIM] = 0;
int dim = 2;            /* define dimension of matrix to be solved */
int n = 0;              /* define step , and initialize it */
float t = STEP_SIZE;                    /* define time step size */

float a1();
float a2();
float b1();
float a3();
float a4();
float b2();
float a5();
float a6();
float b3();
float a7();
float a8();
float b4();
get_ej();
get_ei();
int f_delta();
get_ej_bound(); /* NOTICE NO get_ei_bound */
h1();
h2();
h3();
get_ei_0();     /* at circle point */
float avg_ei();
float one8();
float eps_x[EPS][J_MAX],eps_y[EPS][J_MAX]; /* NEED not I_MAX */
float r[I_MAX][J_MAX];
float theta[J_MAX];

/* external definitions in "util.h" */
float delta_a();
float delta_i();
```

Apr 1 13:14 1987 main.h Page 2

```c
float delta_j();
float delta_l_i();
float delta_l_j();
float eps_alpha();
float eps_beta();
float eps_lamda();
float eps_lu();
float eps_x_i();
float eps_x_j();
float eps_y_i();
float eps_y_j();
float lowerr();
perror();
float upperr();
get_h_0();
readprop();
readmatrix();
readgrid();
dump_h();
dump_e();
copy_e();
copy_h();
e_field();
h_field();  .
prop();
grid();
float addt(),subt();                                /* get pos theta value */

FILE *fin, *fout, *fopen();                         /* I/O file definition */
char outfile[15];                                   /* define output file name */

/* define two Jx Jy functions */
float funcjx(), funcjy();
/* define functions in lusolver.c */
lusolver();
exchange();
test();
Init_file();                                        /* open files routine */
strcpy();                                   /* utility routine */

/*====================== for statistics =====================*/
#define NUMSTAT 10
float ei_stat[NUMSTAT][JJ] = 0;
float ej_stat[NUMSTAT][JJ] = 0;
e_field_stat();
e_field_stat_pr();
headpr();

int half_energy[10];                        /* energy half to be printed */
int int_energy[10];                         /* energy to be printed */
```

Apr 1 13:14 1987 util.h Page 1

```c
/**********************************************************************
In this head file , I include all utility defined global variables
This is a smallest set of global variables, and will be called by
other big head files.
"util.h"
***********************************************************************/

#include <stdio.h>
#include <math.h>

#define TRUE 1
#define FALSE 0
#define CIRCLE_DEPTH 5
#define MM CIRCLE_DEPTH                         /* R1 */
#define NN 2*CIRCLE_DEPTH                       /* R2 */
#define II NN+10                                /* R3 */
#define JJ 32                /* theta */
#define EPS II + 6 /* ensure no error */
#define I_START 0                               /* when read material prop */
#define I_END CIRCLE_DEPTH
#define J_MAX JJ + 5 /* ensure that array cause NO error */
#define I_MAX JJ+10 /* ensure that array cause NO error */

extern float eps_x[][J_MAX],eps_y[][J_MAX],r[][J_MAX];
extern float theta[];

extern float delta_a();
extern float delta_i();
extern float delta_j();
extern float delta_l_i();
extern float delta_l_j();
extern float eps_alpha();
extern float eps_beta();
extern float eps_lamda();
extern float eps_lu();
extern float eps_x_i();
extern float eps_x_j();
extern float eps_y_i();
extern float eps_y_j();
extern float lowerr();
extern perror();
extern float upperr();
extern float addt(),subt();

extern int n;                     /* time level */
extern float t;                   /* time step size */
```

END
DATE
FILMD
3-88
DTIC